BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2019 Bc. Lukáš Kratochvíla

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

VISUAL OBJECT TRACKING IN REALTIME
TRASOVÁNÍ OBJEKTU V REÁLNÉM ČASE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Lukáš Kratochvíla**
AUTOR PRÁCE

SUPERVISOR — **Ing. Karel Horák, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Cybernetics, Control and Measurements**
Department of Control and Instrumentation

Student: Bc. Lukáš Kratochvíla ***ID:*** 171242

Year of study: 2 ***Academic year:*** 2018/19

TITLE OF THESIS:

Visual Object Tracking in Realtime

INSTRUCTION:

General object tracking is used in biological tasks, robotic systems, object localisation in industry and elsewhere.

1. Study general object tracking methods in general (unknown) scene.
2. Generate training and verification datasets.
3. Implement selected method using machine (deep) learning and test on available benchmark.
4. Re-implement selected method into an convenient embedded device.
5. Evaluate implementation and compose technical thesis in English language.

RECOMMENDED LITERATURE:

1. Goodfellow I., Bengio Y., Courville A.: Deep Learning. MIT Press, 2016. ISBN 9780262035613. (deeplearningbook.org)

2. Syelinski, R.: Computer Vision: Algorithms and Applications. Springer,2010. ISBN 978-1848829343.

Date of project specification: 4.2.2019 ***Deadline for submission:*** 5.8.2019

Supervisor: Ing. Karel Horák, Ph.D.
Consultant:

doc. Ing. Václav Jirsík, CSc.
Subject Council chairman

ABSTRACT

Tracking a general object in real time on an embedded device is a challenge. Many algorithms for tracking objects have been developed. This thesis introduces some of these algorithms. Different approaches are discussed, including deep learning. Object representations, evaluations datasets and metrics are shown. Many tracking algorithms are presented, eight of them are implemented and evaluated on the VOT dataset.

KEYWORDS

Object tracking, Embedded device, Real-time tracking, Deep learning, General tracking algorithm, Object representation, Computer vision, Tracking datasets

ABSTRAKT

Sledování obecného objektu na zařízení s omezenými prostředky v reálném čase je obtížné. Mnoho algoritmů věnujících se této problematice již existuje. V této práci se s nimi seznámíme. Různé přístupy k této problematice jsou diskutovány včetně hlubokého učení. Představeny jsou reprezentace objektu, datasety i metriky pro vyhodnocování. Mnoho sledovacích algorimů je představeno, osm z nich je implementováno a vyhodnoceno na VOT datasetu.

KLÍČOVÁ SLOVA

Sledování objektu, Vestavěné zařízení, Sledování v reálném čase, Hluboké učení, Obecné sledovací algoritmy, Reprezentace objektu, Počítačové vidění, Datasety pro sledování

KRATOCHVÍLA, Lukáš. *Visual Tracking in Realtime*. Brno, Rok, 70 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation. Advised by Ing. Karel Horák, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Úvod

Tato práce se zabývá tématem sledování obecného objektu v neznámé scéně v reálném čase. Klade si tyto cíle: přiblížit problematiku, implementovat algoritmy, znovu implementovat algoritmy do vestavěného zařízení, sestavit trénovací, testovací dataset a vyhodnotit výsledky.

Sledování obecného objektu není nová problematika. Tomuto úkolu se počítačové vidění věnuje už od svého vzniku. Řešení by se dalo využít pro celou řadu problémů od aplikací v průmyslu, např. sledování výrobků, přes aplikace ve vědě, např. sledování jednobuněčných organismů, až po aplikace v komerční sféře, např. autonomní vozidla. Třetí možnost, komerční sféra, nabízí největší uplatnění těchto algoritmů. Zvláště v této sféře se setkáváme s obecnou scénou. To je taková scéna, kterou nejsme schopni ovlivnit. V takovém případě potřebujeme algoritmus robustní, který by problematiku zvládl.

Pro sledování objektu je důležité správně zvolit reprezentaci objektu. Algoritmy používané pro sledování objektu obvykle mají tyto 3 části: reprezentace snímku, model objektu a model pohybu. Jako reprezentaci snímku rozumíme úpravu snímku do jiného, tzv. příznakového prostoru. Není však nutné, aby algoritmus měl všechny tři části, např. algoritmus GOTURN nemění reprezentaci snímku a pracuje přímo se snímkem.

Popis řešení

Pro prezentované cíle byl navržený postup: seznámit se s problematikou, popsat algoritmy schopné tuto problematiku uskutečnit, vybrané algoritmy implementovat a vyhodnotit.

První kapitola obsahuje základy z počítačového vidění. Zvláště se zaměřuje na vytváření příznakového prostoru. Také představuje koncepty hlubokého učení, konvolučních neuronových sítí a učení pomocí soustavy modelů.

Ve druhé kapitole se přesouvá zájem k návrhu sledovacího algoritmu. Jsou představeny předpoklady a omezení, která jsou využívána při sledování objektu. Kapitola se také věnuje reprezentaci objektu, postupům jak problematiku řešit a několika algoritmům, které jsou popsány podrobněji.

Další kapitola se zabývá datasety, metrikami a dostatečně výkonnými vestavěnými zařízeními. Jednotlivé datasety stručně popisuje a více se věnuje metrikám. Vestavěná zařízení jsou chápána jako jedno-deskové počítače schopné provozovat Operační systém. Součástí jsou specifikace a vzhled jednotlivých zařízení.

Praktická část práce je prezentována v předposlední kapitole. Také jsou prezentovány zvolené algoritmy, zařízení a dataset, na kterém bylo provedeno vyhodnocení. Podrobněji je popsána implementace programů. Poslední kapitolou je shrnutí a možnosti budoucího směřování práce.

Zhodnocení výsledků

Zvolené algoritmy AOB, MIL, KCF, TLD, MOSSE, Median Flow, GOTURN a CSR-DCF byly úspěšně implementovány i znovu implementovány do vestavěného zařízení Raspberry Pi. Byl sestaven trénovací dataset i navrženo jeho rozdělení pro vytvoření validačního datasetu. Implementované algoritmy byly vyhodnoceny na VOT2014 datasetu. Z výsledků vyplývá, že z implementovaných algoritmů jsou pouze *dva* schopné pracovat v reálném čase na vestavěném zařízení. Konkrétně Median Flow a MOSSE. První jmenovaný byl vyhodnocen jako nejvhodnější kandidát a to i přesto, že dosáhl nižší úrovně snímků za vteřinu, a to protože má menší chybovost. Také byly srovnány dvě různé implementace algoritmu GOTURN. Originální implementace dosáhla lepších výsledků. Podrobné výsledky hodnocení mohou být nalezeny v přílohách. Implementace byla provedena pomocí programů, které jsou přiložené na vloženém CD.

DECLARATION

I declare that I have written the Master's Thesis titled "Visual Tracking in Realtime" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

My endless thanks belong to my supervisor Mr. Ing. Karel Horák, Ph.D. for his time, consultations, patient and important suggestions for my work. Then I want to thank my friend Florian Wintel, my cousin Pavla Kratochvílová, my beautiful girlfriend and all who help me to correct my thesis. Last but not least I want to thank my family for everyday support and understanding in difficult parts of my studies as well of my live.

Brno ....

author's signature

Contents

List of Figures

List of Tables

Listings

Introduction

Object tracking received more attention in recent years. Therefore, wide progress was made in this field. Strong motivations for this work are challenges, for example Visual Object Tracking or Multiple Object Tracking. Another motivation is the use of computer vision, for example in self-driving cars. This task belongs to the computer vision field. This field is quite new, but quickly progressing. The object tracking can be used in all kind of environments from industry application, through application in science to commercial space. The last one is usually the most challenging. In the case of industry or science, we usually can modify the environment to our goal. The definition of the general scene is that we cannot change it.

This thesis investigates object tracking in real-time on an embedded device and is divided into several parts according to goals: understanding the problematic, implementation and reimplementation of the algorithms in embedded device, creating the dataset and evaluation.

In the first chapter, computer vision basics are presented. The focus is on the creating of the feature space. Concepts deep learning, convolution neural networks, and ensemble learning are presented.

The second chapter changes the focus to the design of the tracking algorithm. The constraints and assumptions, related to object tracking, are presented. The chapter also discusses object representations, tracking approaches, and some algorithms are detailing described.

The next chapter presents datasets, metrics, and embedded devices. The datasets are briefly described and more space is dedicated to metrics. The possible embedded device are presented. This kind of device is very powerful today. It is called single-board computers because even an Operating System (OS) can run on it.

The practical part is presented in the penultimate chapter. Firstly, the implementation is described. The reimplementation follows. Results, training and validation dataset are provided and discussed. The last chapter is a conclusion and the future direction of the work.

1 Theoretical Background

The traditional approach is to build features from a picture by finding locally significant pixels and detect motion. The object is then tracked based on these features. We can find features with different approaches. We will go through some basic background to these approaches. We will also mention the new Deep learning approach.

1.1 Corner and Edge

Corner and edge detection is the base detection task. The reason for searching corners and edges is that these points are spatially significant, which means that we can simply see their motion. The methods for solving this problem are called *detectors*.

Basic detectors are Harris and Shi&Tomasi. Both use a matrix of brightness change, but each uses a different metric. If we want unambiguous points, we can use a small neighbourhood. Then we create a descriptor, e.g. SIFT or SURF. Fig. 1.1 shows a detection using the SIFT method.

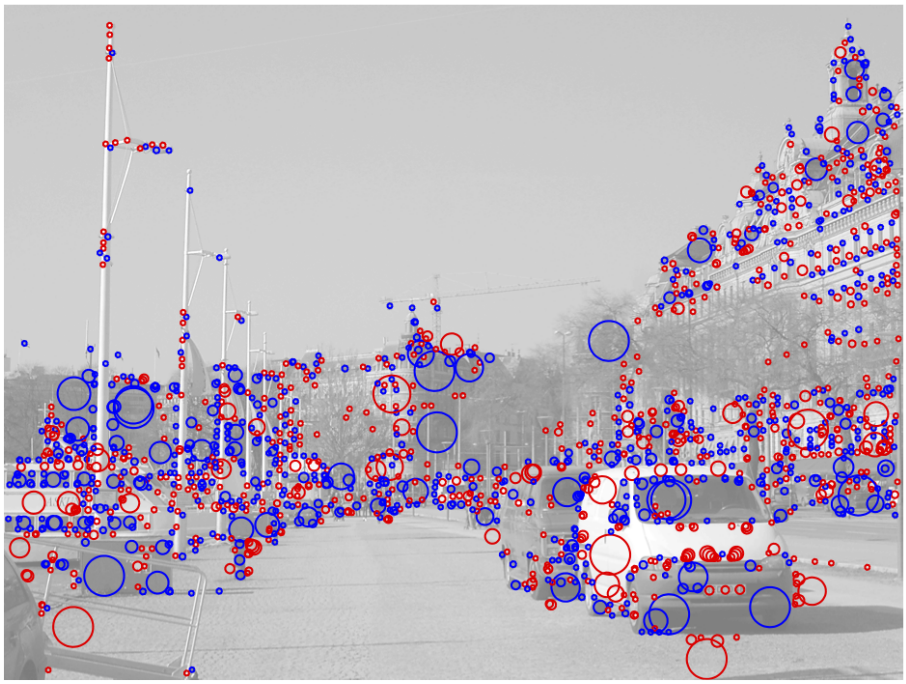

Fig. 1.1: Detection of significant points by SIFT method. The image was published in article Scale Invariant Feature Transform by T. Lindeberg. [online] Scholarpedia, 2012, [cit. 24.1.2019]. Available from URL: <`http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform`>.

1.2 Optical Flow

Optical flow is a method which calculates motion change and direction for every pixel. Mostly we use it only for Region of Interest (RoI). The basic idea is to approximate image change by the first order Taylor polynomial approximation and ignoring higher-orders. This approach is presented in equation 1.1.

$$f(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t) = \tag{1.1}$$

$$f(x, y, t) + \frac{\partial f}{\partial x}\mathrm{d}x + \frac{\partial f}{\partial y}\mathrm{d}y + \frac{\partial f}{\partial t}\mathrm{d}t = \tag{1.2}$$

$$f(x, y, t) + f_x\mathrm{d}x + f_y\mathrm{d}y + f_t\mathrm{d}t \tag{1.3}$$

When we assume that the equation 1.1 is equal to zero, we obtain the time change dependence on image change and derivation of the spatial coordinates, this is shown in equation 1.4.

$$f_x\mathrm{d}x + f_y\mathrm{d}y + f_t\mathrm{d}t = 0 \rightarrow -f_t = f_x\frac{\mathrm{d}x}{\mathrm{d}t} + f_y\frac{\mathrm{d}y}{\mathrm{d}t} \tag{1.4}$$

From the equation 1.4, we can define an optical flow relationship, as follow:

$$\mathbf{v} = (\frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d}y}{\mathrm{d}t})^T \tag{1.5}$$

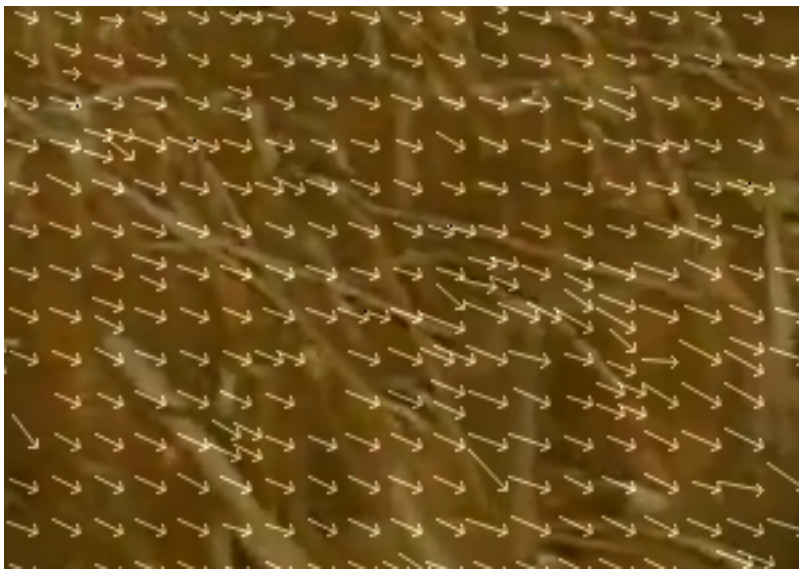

Fig. 1.2: Optical flow example extracted from video frame

1.3 Histogram of Oriented Gradients(HOG)

Another method for feature extraction is HOG. It is based on the illumination gradient method. The gradient is a mathematical term that generalizes the derivative. In this case, it refers to illumination change. Based on this we can create a good feature for tracking. For example, the pedestrian implementation [34] works with image 64 x 128 x 3 (width x height x channels) and generates an output feature vector of length 3780.

Firstly, this descriptor takes the image, takes the patch and resizes it to the mentioned shape. Then it calculates the gradients by derivative filters. At every pixel, the gradient has magnitude and direction. The processed image is divided into 8 x 8 cells. For each cell, HOG is computed. The histogram has 9 bins corresponding to the angles 0, 20, ..., 160. The angles in between are proportionally divided into both bins. The penultimate step is 16 x 16 block normalization. This blocks consist of 4 histograms of 9 bins, concatenated into a 36 x 1 vector. After normalization we have 105 blocks of dimension 16 x 16, where each has a 36 x 1 vector. Thus the final vector has 3780 elements.

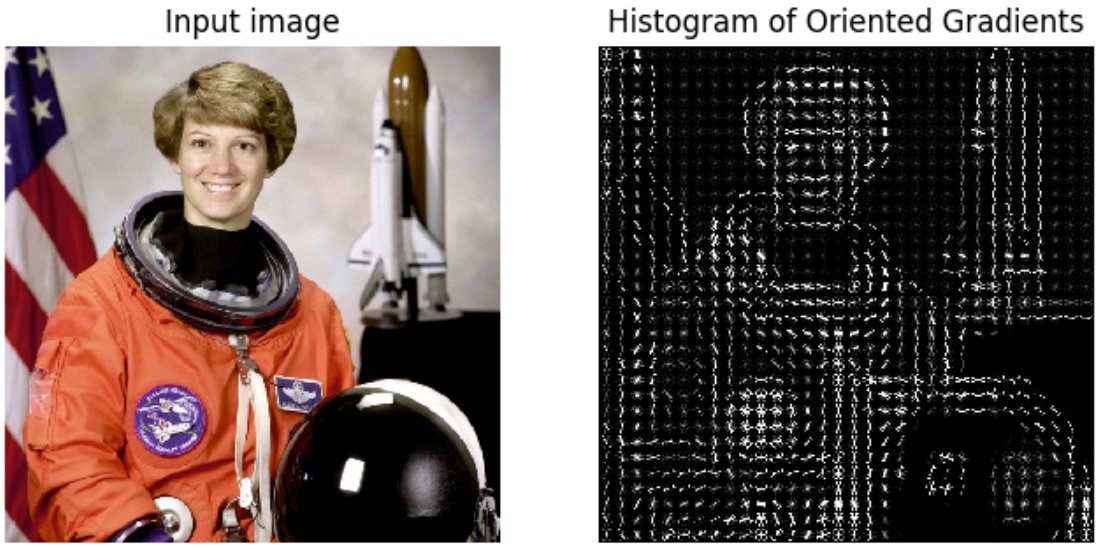

Fig. 1.3: Histogram of Oriented Gradients example. Image published in article Histogram of Oriented Gradients by scikit-image development team. Available from URL: <`https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html`>

1.4 Haar-like Features

The Haar-like features are features inspired by Haar wavelet, the mathematical sequence of a square-integrable function of the unit interval [0, 1], which are used for wavelet analysis. The feature is shown on Fig 1.4. The rectangle features are built from two areas. The white area has value 1 and the gray area has value -1, the sum over the whole feature is 0.

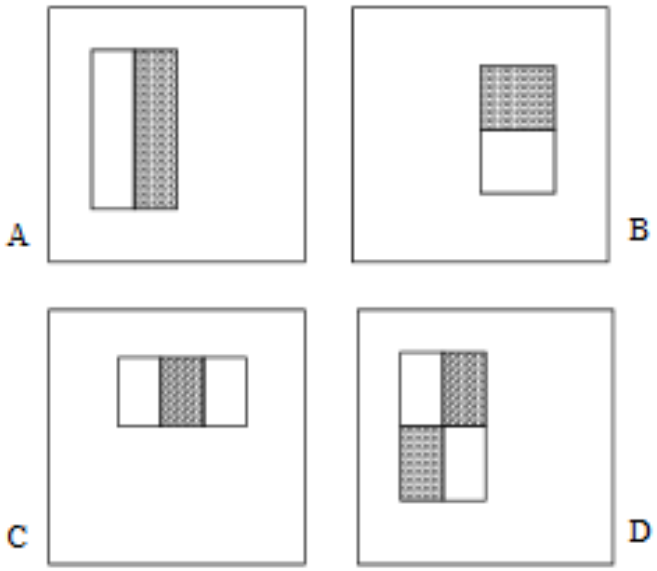

Fig. 1.4: Haar-like features example published in article Rapid Object Detection using a Boosted Cascade of Simple Features by VIOLA, Paul and JONES Michael in 2001. Available from URL: <`https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf`>

This feature can be simply computed from an integral image. The integral image is a summed-area table and can be computed as follows:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{1.6}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{1.7}$$

where the $i(x, y)$ is the original image, the $ii(x, y)$ is the integral image, the $s(x, y)$ is the cumulative row sum, $s(x, -1) = 0$ and $ii(-1, y) = 0$.

1.5 Deep Learning and CNN

Recently, the research went a new way. In contrast to the traditional approach, Deep Learning attempts to use a huge amount of data for training the model. Then it uses the trained model for the task. The most time-consuming part is the training part. The running process of the model can be real-time depending on the hardware.

Especially in object detection, this approach is State of the Art. This is caused by huge developments in this field in recent years. This approach can be used for object tracking as is shown in section 2.3.8.

1.5.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is one of the machine learning models. The base of this model is a neural network. The neural network is a model of our biological brain. In the case of CNN, the image is processed by convolution layers. The architecture can be seen in Fig. 1.5. The output from the convolution layer is called *feature maps*. The feature maps correspond with a pattern, which the convolution layer tries to find. When we go deeper, the CNN searches for a more complex pattern.

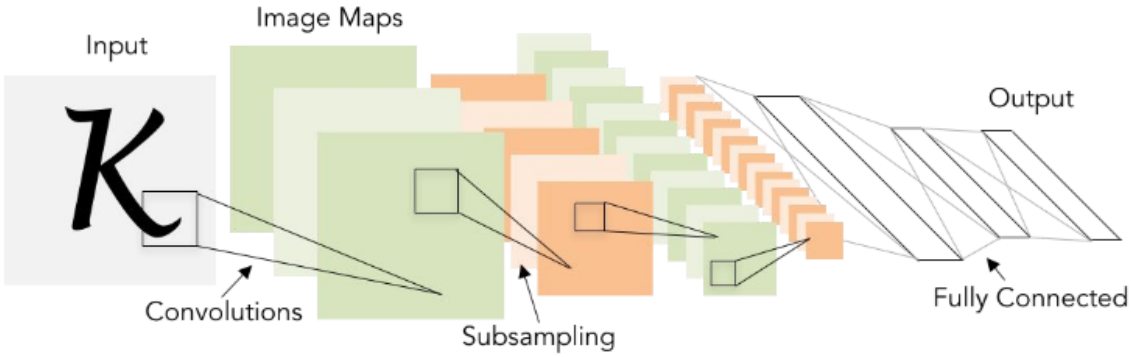

Fig. 1.5: Convolutional Neural Network architecture published in Lecture 5: Convolutional Neural Networks by Fei-Fei Li, Justin Johnson, and Serena Yeung.[online][cit. 19.7.2019]. Available from URL: <`http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf`>.

1.5.2 Back-propagation

The core algorithm responsible for training neural networks is back-propagation. This algorithm determines the way how to set the weights of the model. The algorithm starts from the output layer and goes backwards through the preceding

layers. When we consider weight w, input vector x, activation function f and δ_k output-target difference we can write the rule as follows:

$$w_{j,i} := w_{j,i} - \alpha \sum_k (\delta_k \cdot w_{k,j}) \cdot f'(z_j) \cdot x_i \tag{1.8}$$

1.5.3 Frameworks

For Deep Learning, new frameworks, which allow the use of CNN, were developed. Tensorflow and Pytorch are the two most frequently used. These frameworks implement a neural network structure and provide an environment for code debugging. Other frameworks include Caffe, Darknet, CNTK, MXNet, Chainer, or Keras. For better performance, especially in terms of speed, these frameworks support Graphics Processing Unit (GPU).

1.6 Ensemble Learning

This paradigm works with the idea that in a crowd is a hidden power. The *classifier* is understood as an entity, which tries to distinguish between two classes. As a output returns the predicted class. Methods create sets of classifiers called weak learners and train them on data. A weak learner is a classifier, slightly better than random guess (random guess successful in 50% of classifications). From the set of weak classifiers, ensemble learning creates one strong classifier. Three main steps are shown in Fig 1.6.

Boosting is a method of building strong classifier from weak classifiers. We have a weight for each example. Each boosting round we learn a new classifier. The classifier returns the result (class) to each example. Wrong classified examples get a higher weight for the next learning step. Classifiers which obtain low training errors get higher weights for computing a strong classifier. Adaboost is a special boosting example, which uses sequential combination of weak classifiers.

Bootstrapping is a method of creating more data. It is sensitive to data distribution, data size, i.e. small dataset or outliers can impact the performance. The process can be diverse, but in general, works as follows: generate data samples with replacement from the original data and compute bootstrap data. The bootstrap distribution will differ slightly from the original data.

Bagging is an aggregation of bootstrap distribution. So we first create the bootstrap datasets and then teach a classifier on them. When we have many classifiers, we combine them, for example by using a method called voting. The Simple Majority Voting method is shown on equation 1.9.

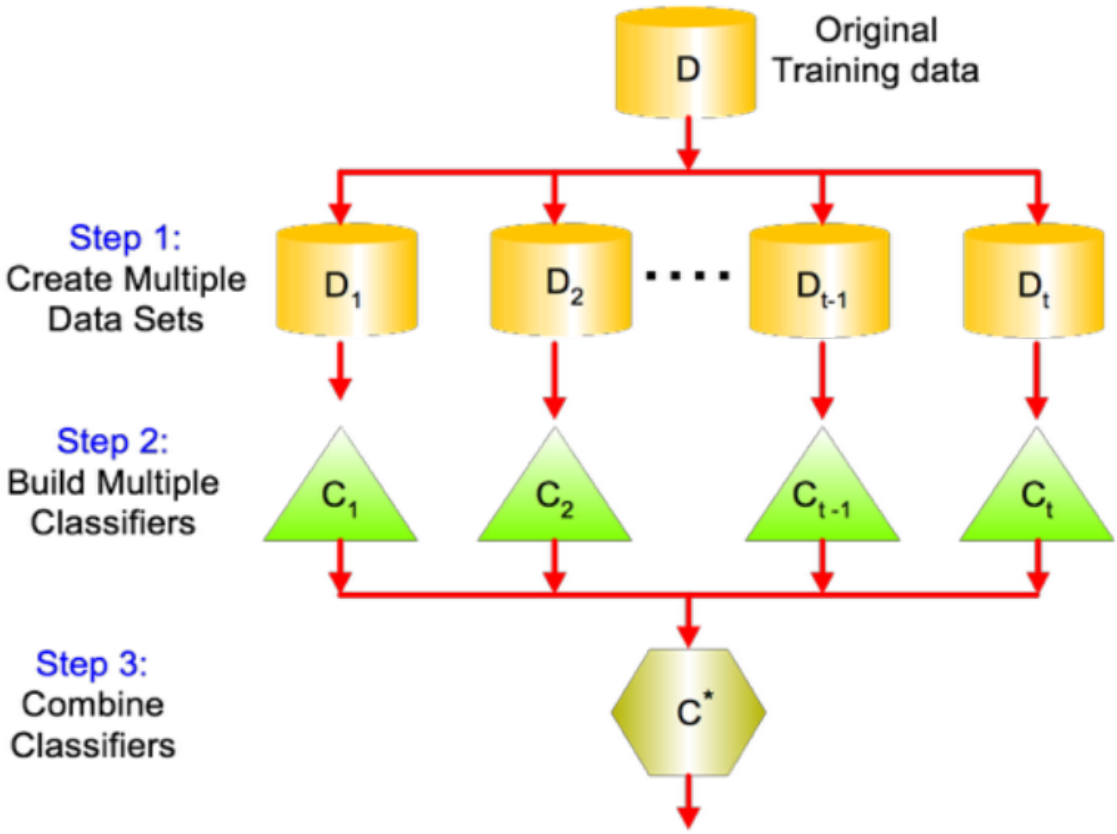

Fig. 1.6: In ensemble learning overview are shown three main steps: Create multiple datasets, build multiple classifiers, and combine classifiers. [44]

$$V_j = \sum_{t=1}^{T} v_{t,j}, j = 1, 2, \ldots, C \tag{1.9}$$

where $v_{t,j}$ is a vote by weak learner for class j (0,1), C is a number of classes and T is a number of weak learners. [14]

2 Design of Tracking Method

2.1 Constraints and Assumptions

Object tracking could be seen as easier than object detection because we can use the following constraints.

2.1.1 Motion Constraints

Motion constraints simplify the search problem. We use constraints for tracking a rigid body. The three most significant ones are shown on Fig 2.1.

- maximal velocity(a) - constrains the area of the object's location
- maximal acceleration(b) - given by the weight of the object
- common motion(c) - topology of rigid object does not change

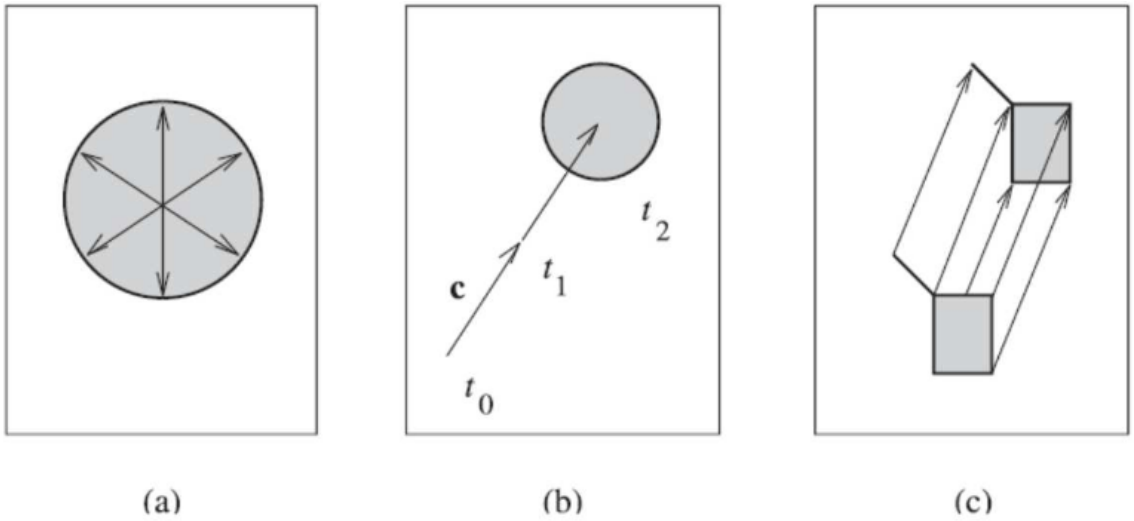

Fig. 2.1: Constraints to simplify the motion of rigid body [5]

2.1.2 Object Representations

Object representation is a key part of the searching algorithm. Fig 2.2 illustrates many different object representations. The main difference between the various representations is computational complexity. Representation is connected to a task, which we want to achieve, e.g. for segmentation, we want the silhouette object representation.

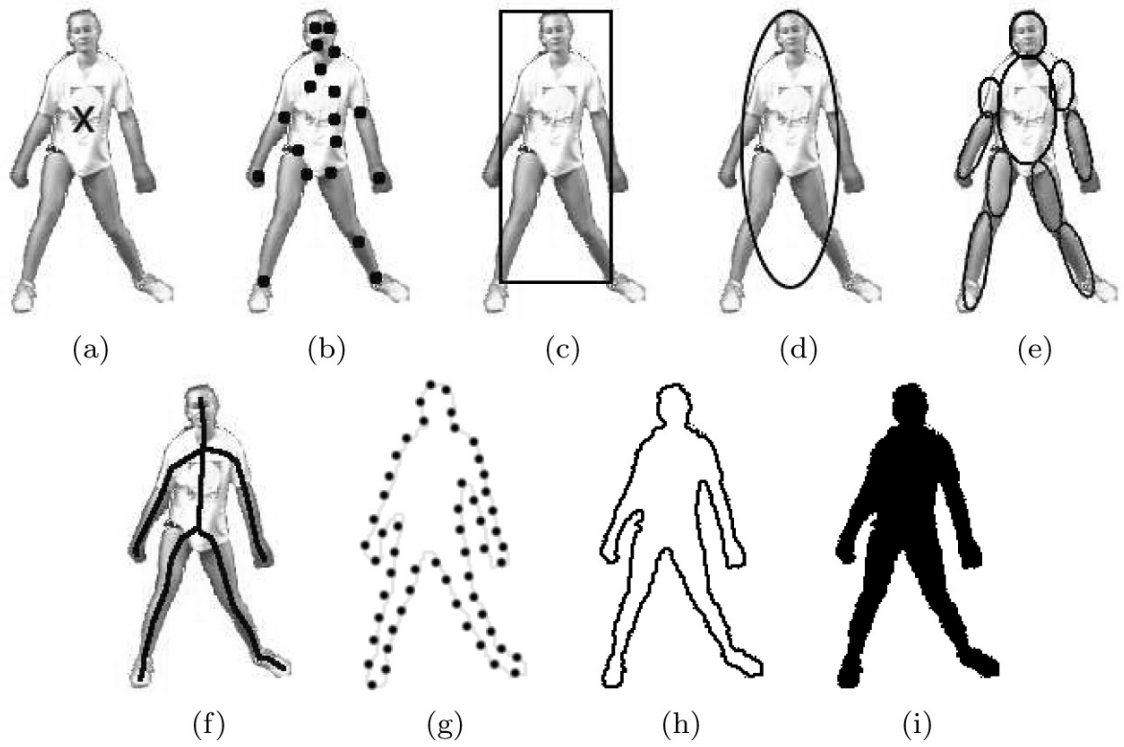

Fig. 2.2: Object representation examples: (a) Centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object sceleton, (g) control points on object contour, (h) complete object contour, (i) object silhouette. [1]

Point or Multiple Points

The simplest object representation form is an approximation by a point or multiple points. This representation is useful for small objects and has the advantage of small computational complexity. For tracking, this representation requires solving the correspondence problem. This problem is finding the same point in the next frame. It is used for example to combine images to build a panoramic image. Especially when tracking multiple points, the task can be quite difficult.

Primitive Geometric Shape

The most frequently used representation is a primitive geometric shape. The advantage of this representation is simplicity and small computational complexity. A disadvantage could be the too simple representation, i.e. patch can include non-object pixels and, conversely, some object pixels can be excluded from the patch.

Articulated Shape Model

When we want to use more prior information, we can create a slightly more complicated model called articulate shape model. This model consists of basic patches and joints. The relationships between patches are defined by kinematic equations. Joints are connections between patches that allow moving patches constrained by these kinematic equations.

Silhouette

The segmentation task is focused on finding a silhouette. The silhouette is an object representation that includes all pixels of the object. This is the ideal which we want to achieve. This problem is computationally expensive.

2.1.3 Short-time vs. Long-time Tracker

The whole task is based on searching for a target in a new frame. As an input, we have the previous frame, a bounding box around the target and a new frame. The output is the target's position, i.e. bounding box in the new frame.

Short-time tracker is not assumed to be capable of re-detection of the target. Thus once the target is lost, the short-term tracker will probably not be able to recover, e.g. the GOTURN tracker.

Long-time tracker is an extension of the short-time tracker. It usually has a part responsible for detecting lost the target and a detection part which tries to find the target again. The tracker should be able to run for an infinite time, e.g. the TLD tracker.

2.2 Related Approaches

The object tracking task is defined by the target's position in a previous image and a new image. The tracking algorithm tries to predict the new position of the target. On Fig 2.3, we can see the main tracking method categories and their relationships. In Tab 2.1 representative methods for each category are shown.

2.2.1 Point Tracking

The point tracking approach can be divided into two categories: deterministic and stochastic. The goal is to find the same point in the next frame. In this method, a point representation is used.

Tab. 2.1: Representative work in a field [1]

Representation	**Categories**	**Representative work**
Point tracking	Deterministic	MGE tracker [6] GOA tracker [8]
	Stochastic	Kalman filter [9] JPDAF [10] PMHT [11]
Tracking single model	Model matching	Mean-shift [12] KLT [13] Layering [15]
	More views tracking	Eigentracking [16] SVM tracker [17]
Silhouette tracking	Template matching	Hausdorff [18] Hough transform [19] Histogram [20]
	New contour inference	State space models [21] Variational methods [22] Heuristic methods [23]

The deterministic methods solve the correspondence problem as an optimizing problem. Deterministic methods use optimizing methods, such as the Hungarian method or some greedy algorithm.

The stochastic methods are based on a recursive optimizing process. There are two types of methods: Kalman filter and sequential Monte Carlo method, for example Partial filter.

Greedy Algorithms These algorithms try to solve the problem based on a locally best decision and do not revise their decision later. [7] An example of a greedy algorithm is Greedy Optimal Assignment.

Kalman Filter Kalman filter is an algorithm based on prior linear model guess and its posterior correction. The correction is calculated by model feedback. The constraint is the assumption, that the model state has a Gaussian decomposition.[1]

Partial Filter Partial filter is based on computing conditional state density

$$p(X_t|Z_t)$$

in time t from examples and their weights. [24]

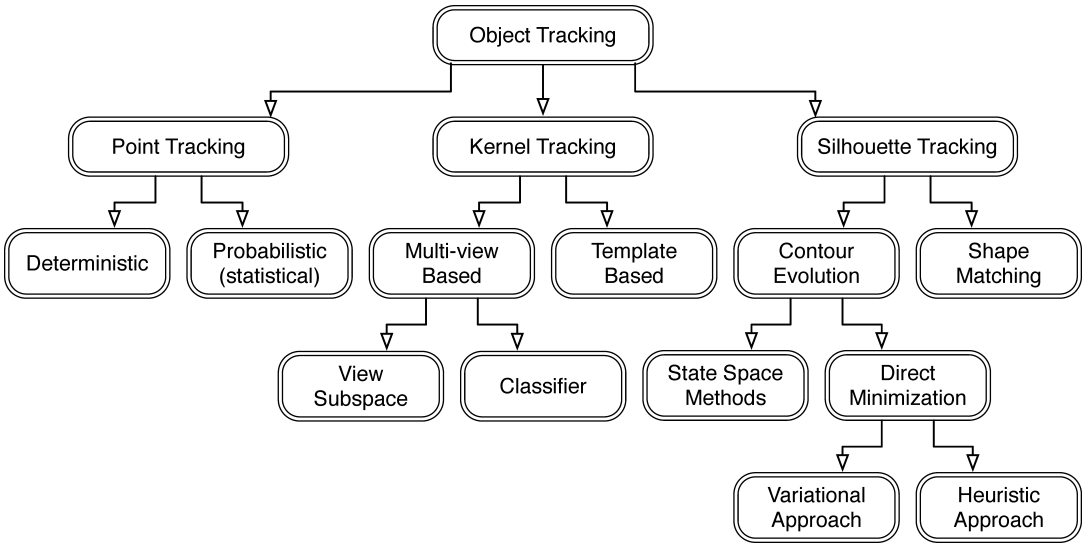

Fig. 2.3: Division of tracking algorithm by used approach [1]

2.2.2 Single Model Tracking

Single model tracking is used the most widely, because of the object representation. A simple model can be a rectangle called a *bounding box*. The first approach how we can try to find our model in the image is model matching. Or we can try to learn the context around the object. The second approach called *more views tracking* tries to learn more context information. The bounding box is not the only model used, we can also use the articulated shape model, or a composition of simple models. The analysed motion is tracked w.r.t. affinity transform, i.e. scope change, rotation or transition displacement.

Model matching is a method for finding the model in an image. As a model we understand a model representation as was mentioned before, e.g. histogram illumination inside the bounding box. [12]

The more views tracking method is based on the assumption that object can significantly change with motion. Simple model matching can have a problem with this situation.

One example is to use *eigenvector space*. The eigenvector space is a mathematical concept used for a description of a linear transformation. This space consists of eigenvectors, non-zero vectors, which can be changed from a linear transformation only scalarly, therefore, they are characteristic for object representation. Based on this space, we create a model template set, which we use for searching. [16] Another method is the algorithm Support Vector Machine. This algorithm tries to distinguish between two classes. One class will be the tracked object and the second class will be the background. [17]

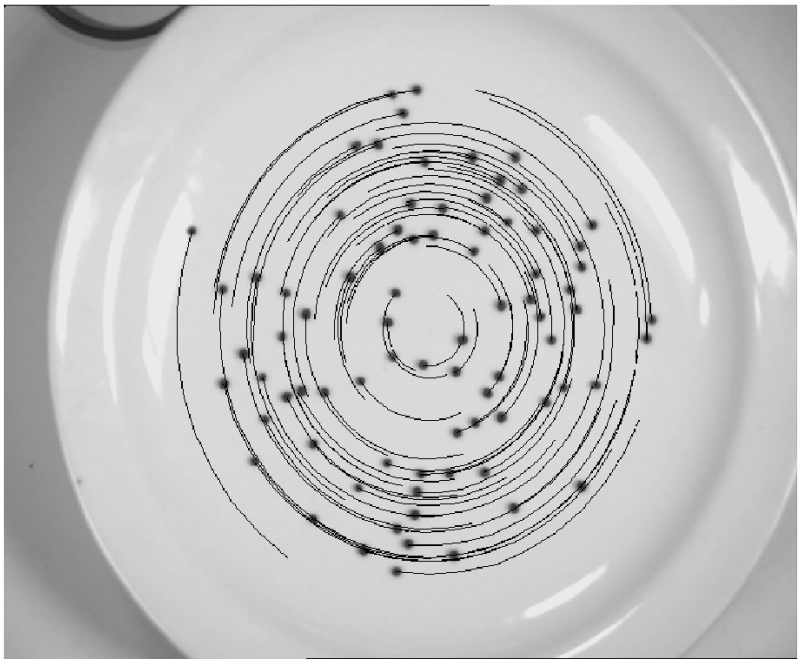

Fig. 2.4: Point detection example on plate by algorithm GOA. [8]

2.2.3 Silhouette Tracking

Silhouette tracking can be divided into two categories: one is template matching and the other is new contour inference. The representation can be created by control points, contour or the whole silhouette.

For template matching, we can use Hausdorff distance. In equation 2.1, A and B are set, which we compare. First we compute the distance for each point from the set A to each point from the set B. For measurements we can use L-norm. Because the classic method is sensitive to data noise, we can use a modification that searches the f quantile besides maximal distance as seen in equation 2.2.

$$h(A,B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{2.1}$$

$$h^f(A,B) = f_{a \in A} \min_{b \in B} ||a - b|| \tag{2.2}$$

The new contour inference is provided by an energetic equation. The contour is defined as follows $p_n = [x_n, y_n], for \quad n = 0, 1, ..., N$. The new contour is searched w.r.t. equation 2.3. The equation tries to minimize the function E_s. Where E_N controls contour smoothness, E_I are forces, which affect the contour from inside and E_T are forces from the initial arrangement.

$$E_s = \sum_{n=1}^{N} E_N\{p_n\} + \sum_{n=1}^{N} E_I\{p_n\} + \sum_{n=1}^{N} E_T\{p_n\} \tag{2.3}$$

2.3 Algorithm Comparison

In this section will be introduced eight algorithms capable of tracking objects in real time. The first two create features space and apply a template matching. Next three are based on correlation filters and *Fast Fourier Transform*. This approach brings fast performance. Median Flow and Tracking, Learning, and Detection are connected too. TLD use The Median Flow algorithm for tracking. The last one is a different approach, which uses CNN. GOTURN is the only one, which is trained offline. The rest is learned in an online manner.

2.3.1 Online Adaboost with Haar-like Features (OAB)

OAB is a learning algorithm based on the boosting method and simple Haar-like features. The main quest is to achieve a stable, continuously updated model. The algorithm takes the initialization bounding box as a positive example and several images from the background as negative examples for learning the model. In the initialization, several iterations of the online boosting algorithm are performed until a stable model is achieved. [30]

The tracking step starts by simple template matching. The RoI is evaluated and for each position a confidence value is obtained. The confidence map is analysed and the shift of the target is found. [30]

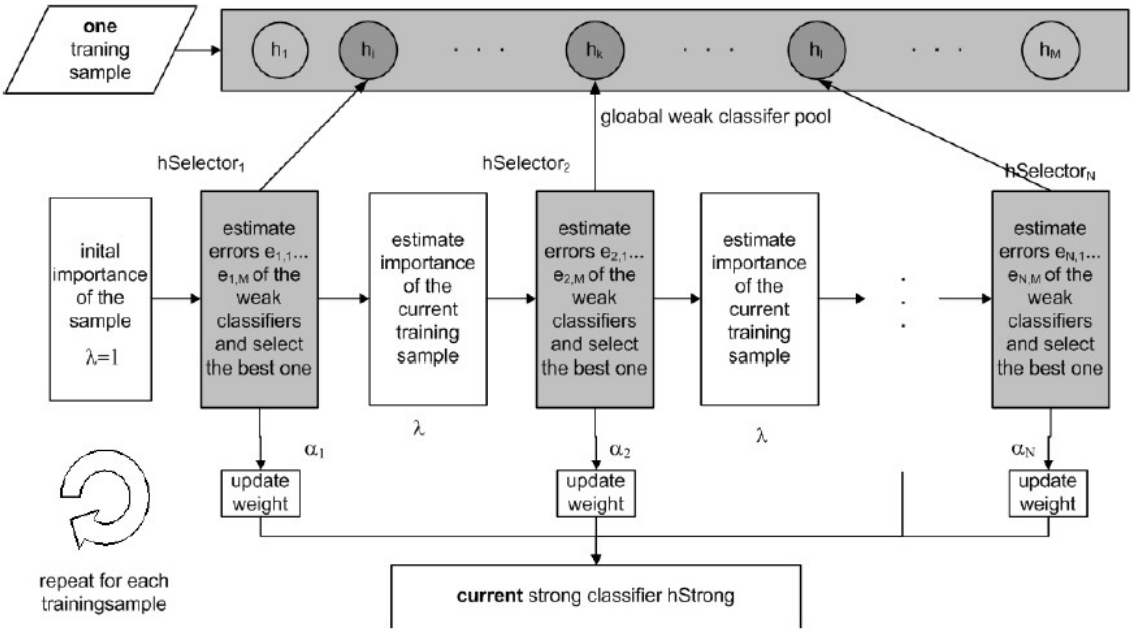

Fig. 2.5: Principle of online boosting [30]

The next step is the Adaboost method, originally proposed in [32] extended of online learning. This method is based on Ensemble Learning, as mentioned in section 1.6. The new part is selectors, which select a hypothesis w.r.t. the optimization criterion. The strong classifier is computed as a linear combination of selectors. The principle is shown on Fig 2.5. [30]

2.3.2 Multiple Instance Learning (MIL)

The Multiple Instance Learning method is based on learning an adaptive appearance model. The algorithm contains 3 main components: image representation, appearance model and motion model. [25]

The OAB algorithm creates positive and negative examples for learning, but does not use them as examples, instead it creates "bags". A positive bag is understood as a set of examples, that contains at least one positive example, but is considered as negative otherwise. For constructing a positive bag the tracker has to bootstrap itself. The comparison of different model updates is shown in Fig 2.6. [25]

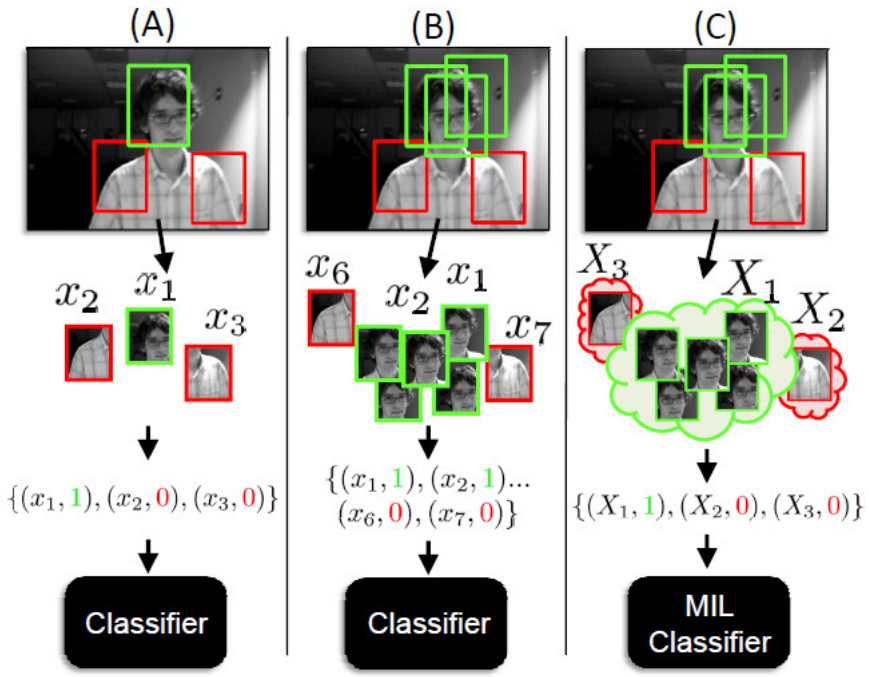

Fig. 2.6: Update a discriminative model: (A) a single positive image patch, (B) several positive image patches, (C) using one positive "bag". [25]

The image representation consists of a set of Haar-like features, constructed for each image path. The appearance model is a discriminative algorithm, which predicts if the object appears in an image patch. As a motion model is assumed

that object can move only to distance s. In the distance s, new images are cropped and the location is upgraded by the *greedy algorithm*:

$$l^* = l(\arg\max_{x \in X^s} p(y = 1|x)) \tag{2.4}$$

where l^* is the new target position and $X^s = \{x|s > ||l(x) - l^*_{t-1}||\}$ is a positive bag. Then a new positive bag $X^r = \{x|r > ||l(x) - l^*_{t-1}||\}$ is created, where $r < s$, and negative bags $X^{r,\beta} = \{x|\beta > ||l(x) - l^*_{t-1}|| > r\}$. The discriminative algorithm is updated with these bags by the MIL-boost method. [25]

2.3.3 Maximum Output Sum of Square Error (MOSSE)

The algorithm is based on cross-correlation. It tries to find the kernel which will have the maximum output for Peak to Sidelobe Ratio (PSR). This ratio measures the strength of correlation peak. It can also be used for occlusion detection. The method tries to deal with small training datasets, learn from one image - find more general relations and generate a stable filter. The advantage of this algorithm is speed because the algorithm uses Fast Fourier transform for correlation computation. [33]

The processed image is cropped around the object from the previous frame and a tracking window is created. The target is tracked by correlating the filter over the tracking window. The target location corresponds to the maximum value in the output of the correlation. For the reduction artefacts in the Fast Fourier transform, the tracking window is preprocessed as follows: the values of pixels are transformed by *log* function, normalized to have mean of 0.0, norm of 1.0, and a cosine window is applied. The correlation process is shown on Fig 2.7.

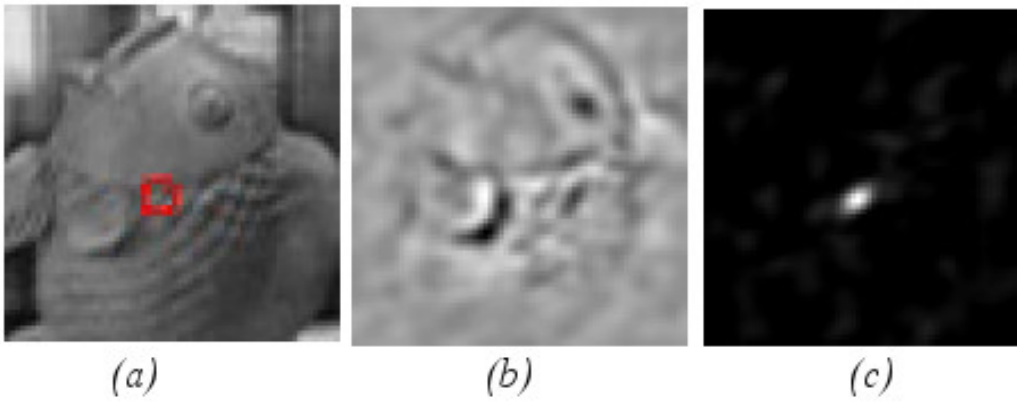

Fig. 2.7: The example of MOSSE detection: (a) is a input image, (b) is a fiter, and (c) is the correlation output. [33]

For the training, we need image and labels. The image will denoted f_i and the label g_i will be an image generated from the ground truth with ($\sigma = 2$)2D Gaussian-shaped peak centred on the target location. The filter, which we want to achieve, is a conjugated counterpart to one on equation 2.5, where the division is element-wise. [33]

$$H_i^* = \frac{G_i}{F_i} \tag{2.5}$$

The minimization problem takes the form:

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{2.6}$$

where $\odot$ is element-wise product. The closed form expression for finding MOSSE filter can be found as:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{2.7}$$

2.3.4 Kernelized Correlation Filters (KCF)

Another discriminant method uses the circular matrix and Fast Fourier transform. The advantage of this algorithm is speed and generalization. It can run at hundreds of FPS. The speed is reached by taking the Fast Fourier transform and using it for correlation. The algorithm uses the kernel matrix, defined later in this paragraph. In the simplest linear case, it collapses to the MOSSE method. For better Fourier analysis results, the edges of the image are updated with a cosine function. [27]

Circulant matrix $C(u)$ is an $n \times n$ matrix obtained from an $n \times 1$ vector u by concatenating all possible cyclic shifts of u:

$$C(x) = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ x_n & x_1 & x_2 & \dots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \dots & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \dots & x_1 \end{bmatrix} \tag{2.8}$$

This matrix is an essential part of this method. An example of using the matrix is shown on Fig 2.8. The goal is to find the function $f(z)$:

$$f(\boldsymbol{z}) = \boldsymbol{w}^T \boldsymbol{z} = \sum_{i=1}^{n} \alpha_i \kappa(\boldsymbol{z}, x_i) \tag{2.9}$$

where κ is the kernel matrix, e.g. the dot product between all pairs of samples, x_i is training example, $\boldsymbol{z}$ is input example and α_i refers to a parameter under optimization. The solution is:

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y} \tag{2.10}$$

where λ is a regularization parameter that controls overfitting, K is the kernel matrix and $\mathbf{y}$ is output. If K is a circulant matrix, the equation 2.10 can be rewritten with properties of the circular matrix as follows:

$$\hat{\boldsymbol{\alpha}} = \frac{\hat{\boldsymbol{y}}}{\hat{\boldsymbol{k}}^{xx} + \lambda} \tag{2.11}$$

where $\hat{}$ denotes a Discrete Fourier transform and $\hat{\boldsymbol{k}}^{xx}$ is the first row of the kernel matrix $K = C(k^{xx})$. The traning image is created similarly as for MOSSE. In OpenCV implementation, a Gaussian kernel is used. [27]

Fig. 2.8: An example of vertical cyclic shifts, which can be performed by a circular matrix. [27]

2.3.5 CSR-DCF

Discriminative Correlation Filter with Channel and Spatial Reliability is a method, which uses a discriminative correlation filter similar to MOSSE or KCF. But the process of creating a filter is different. In addition, it has more channels used for computing the response and it enables a wider spacial search. The channels are assumed to be independent, therefore, only one filter is created. Every frame is processed in two steps: the localization and update step. [31]

The localization step consists of three parts. The first is computing correlation filter responses weighted by channel reliability weights computed for the previous frame. The second step is an estimation of new channel reliability weights. Next, it is estimated scale from a pyramid space. [31]

The update step has more parts. On the begin, it is extracted color histograms of foreground and background and update model histograms. The next step is computing the spatial reliability map (binary mask) used for the constraint correlation filter. The observation for i-th pixel is estimated as follows:

$$p(y_i) = \sum_{j=0}^{1} p(y_i^c | m_i = j) p(y_i^x | m_i = j) p(m_i = j) \tag{2.12}$$

where $p(y_i^c|m_i = j)$, $p(y_i^x|m_i = j)$, and $p(m_i = j)$ are the appearance likelihood, the spatial likelihood and the foreground/background prior probability. The appearance likelihood is computed form extracted histograms. The prior probability is computed from ratio between foreground and background. The spatial likelihood is defined as weak spatial prior, near to center pixel equal to 0.9 and change to a uniform prior away from the center pixel. For the real implementation is necessary to implement Markov random field, because of noisy data. This reliability map is in fact segmentation estimation. Then the new correlation filter is estimated by Lagrangian. The second channel reliability weights are computed and element-wise multiplied with the first one. The last step is a filter and channel weights update. [31]

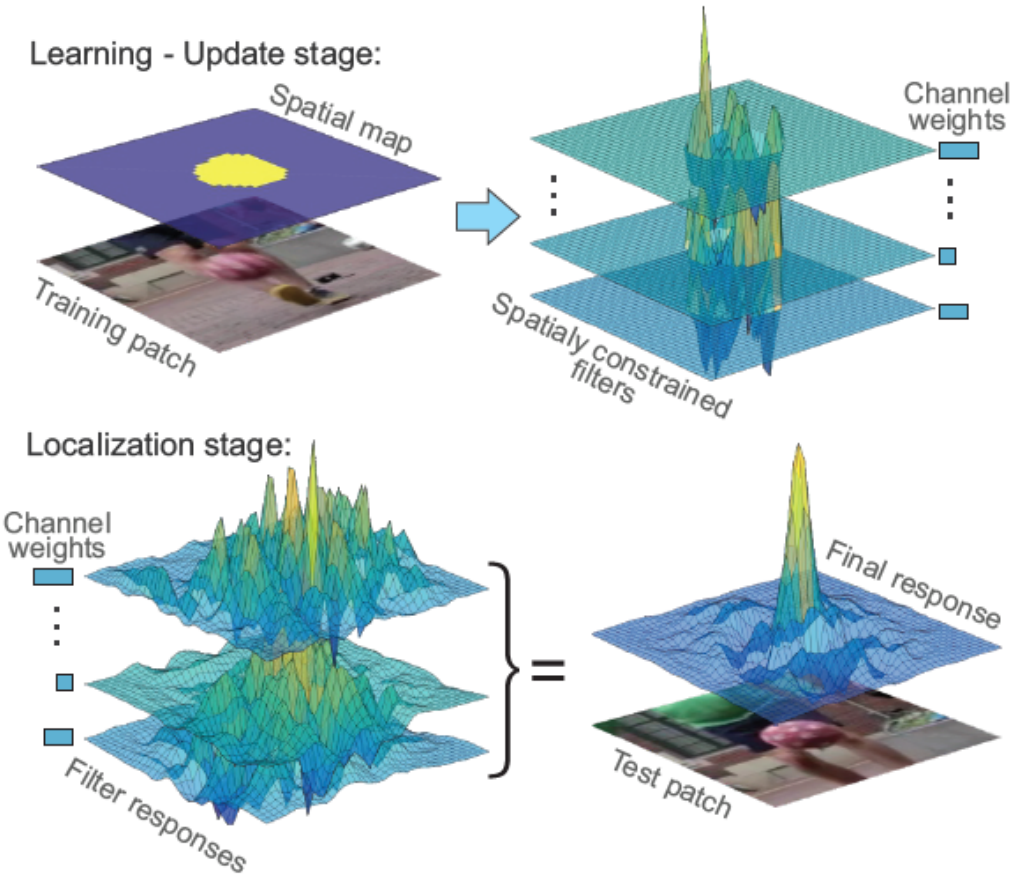

Fig. 2.9: Overview of CSR-DCF approach [31]

2.3.6 Median Flow

The method proposed in [28] is based on the Forward-Backward error and Lucas-Kanade tracker. The algorithm has these steps: initialize points, track points, estimate tracking error, filter out outliers and update bounding box. The overview is shown on Fig 2.10.

For the initialization, a grid in the initial bounding box is created. In this grid,there are points equidistantly created and these points will be tracked. The tracking error is computed by the Forward-Backward process. The worst 50% points are filtered out. The bounding box scale change is defined as the median ratio between the distance of each pair of points, i.e. new versus previous one. The displacement is performed using the median over each spatial dimension of the remaining points. [28]

Forward-Backward error is a novel process for computing tracker reliability. The process consists of two steps. The first step is to predict the output forward through the sequence. The second step is to compute the prediction backward through the sequence. The difference between forward and backward prediction is called Forward-Backward error. [28]

Lucas-Kanade tracker Lucas-Kanade tracker [37] is a method based on optical flow and assumes that brightness intensity is constant. For computing the optical flow 1.5 we also assume that for all pixels in some region the optical flow is the same.

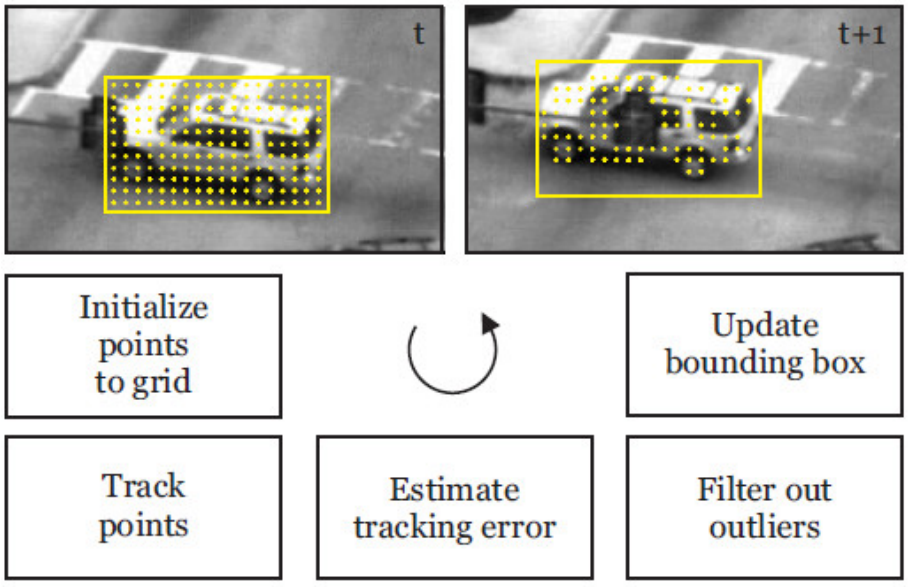

Fig. 2.10: Overview of Median flow approach [28]

2.3.7 Tracking, Learning, and Detection (TLD)

The Tracking, Learning, and Detection framework tries to find a balance between detection and tracking approaches. The main advantage is that the tracker is constructed as long-term tracker, i.e. it should be able to recover from failures. The focus of this method is on these challenges: deal with arbitrary video, never degrade the detector and operate in real time. For this task, the method uses these three states: tracking, detecting, and learning. The three states can be seen in Fig. 2.11.

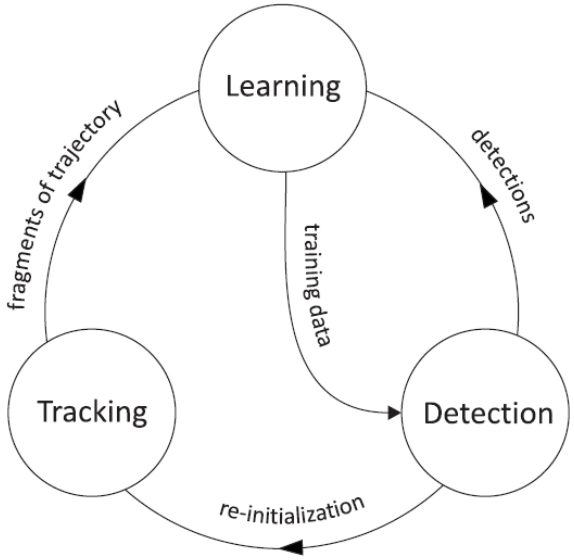

Fig. 2.11: The block diagram of the TLD framework [29]

As tracker, the Median Flow tracker, extended with failure detection, is used, which tries to estimate the object's motion. The failure detection is provided by measurement of displacement of single pixels. Let d_i be a single-pixel displacement, d_m median displacement, then if:

$$median|d_i - d_m| > 10pixels$$

the algorithm will not return the bounding box because the failure will be detected.

The decoder assumes that every frame is independent and performs whole image scanning to localize all appearances that have been learned in the past. For this work, the following three stages are used: patch variance check, ensemble classifier, and nearest neighbour. If and only if the appearance passes all three, it is accepted.[29]

The detectors can make two types of mistakes: false positive and false negative. For the purpose to correct them, the algorithm applies the P-N learning. This learning consists of two experts, which work independently. The P-expert attempts to recognize missed detections and N-expert recognises false detections. [29]

2.3.8 Generic Object Tracking Using Regression Networks

GOTURN is a Deep Learning method based on offline training a CNN. The motivation is to take the advantage of a huge amount of data and build a strong model. The algorithm has two states, training and estimation. The network consists of a convolution part of the CaffeNet and 3 fully connected layers, each with 4096 nodes.[26]

For the training, a pre-trained model, trained on ImageNet, is used. The training consists of two steps. The first one is training on images. The reason for training on images is to train the network on more diverse object appearance. From the image, two samples are cropped. One is centred on the target and a second is shifted. The first one is taken as the previous example. The second step is training on video sequences.[26]

The estimation step firstly crops a new image and the previous image to size $k_1 w \times k_2 h$ around the center point in the previous frame. The k_1, k_2 are constants, which set how much context will be provided to the network. In the original implementation, they are equal to 2. Then the cropped images are pass to the network and as the output, are achieved numbers of the bounding box in the form (x_1, y_1, x_2, y_2), here x_1, y_1 are coordinates of the top-left corner and x_2, y_2 the bottom-right.[26]

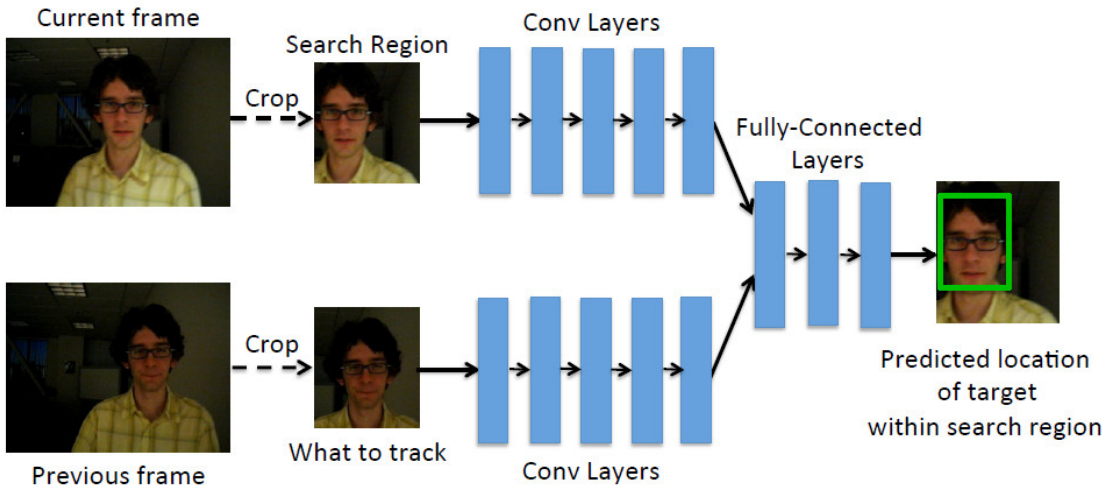

Fig. 2.12: Test image example [26]

3 Evaluation

Next part of the work is focused on available datasets, metrics, which can be used for the evaluation, and the embedded device with enough power for real-time performance.

The tags of the video correspond to attribute, which we want to deal with: Scale Variation (SV), Aspect Ratio Change (ARC), Fast Motion (FM), Low Resolution (LR), Out-of-View (OV), Illumination Variation (IV), Camera Motion (CM), Motion Blur (MB), Background Clutter (BC), Similar Object (SOB), Deformation (DEF), In-Plane-Rotation (IPR), Out-of-Plane Rotation (OPR), Partial Occlusion (POC) and Full Occlusion (FOC). [40]

3.1 Datasets and Challenges

Training data are the most important learning part, but it is quite difficult and expensive to create labelled data. Thanks to challenge-datasets, we do not need to manually create our own datasets. These challenges provide large sets of images, which are necessary for learning algorithms. [40]

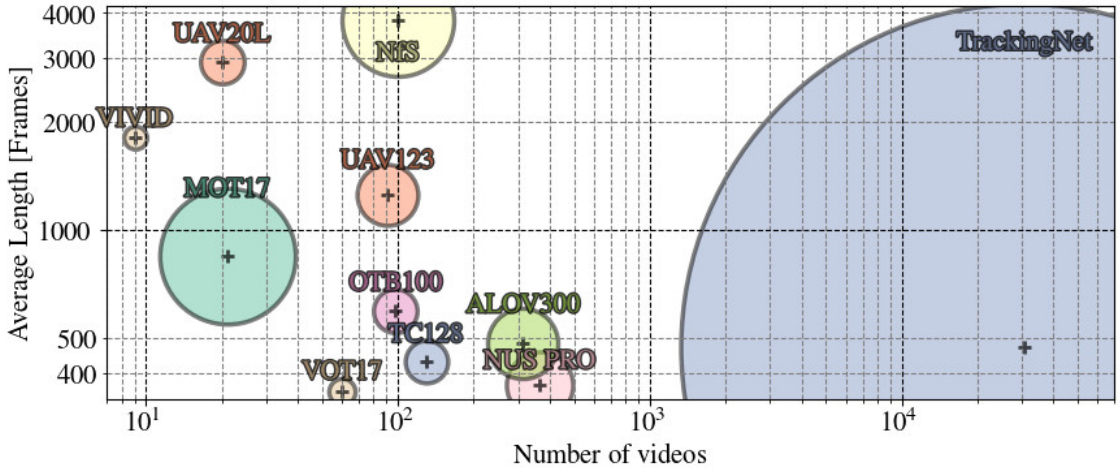

Fig. 3.1: Datasets comparison [40]

3.1.1 Visual Object Tracking (VOT)

The Visual object tracking challenge is concentrated on short-term single object tracking. The challenge started in 2013, and continues to today. The focus is on the short-term tracker. In 2019, five tracking image tasks were introduced by the VOT challenge. [39]

To ensure a common base for evaluation, the VOT challenge authors provide a wide tool-kit for contributing. The whole tool-kit is written in the Matlab/Octave environment. It also provides support for all kinds of different environments like C/C++, or Python.

3.1.2 Multiple Object Tracking (MOT)

This dataset has three main principles: to be publically available, to allow centralized evaluation, and to offer an infrastructure that allows crowdsourcing new data. The first release has 22 sequences with a total of 11,286 frames. [38]

3.1.3 Tracking Net

The Tracking Net is a large-scale dataset and benchmark for object tracking in the wild. It assembles 30,643 videos and 14,431,266 frames in total. [40] The attributes distribution is shown in Fig 3.2.

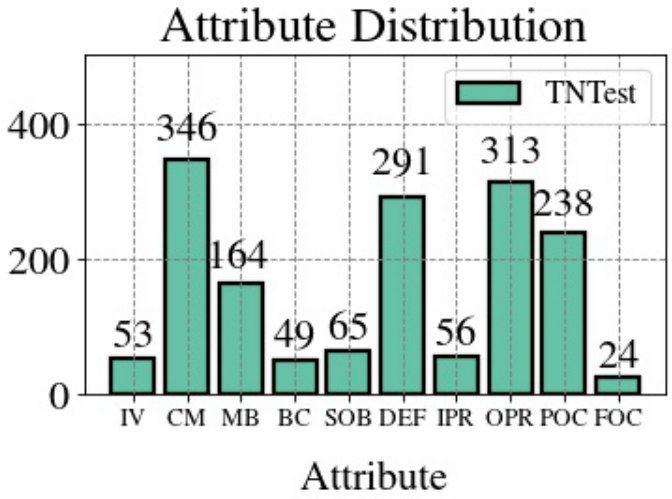

Fig. 3.2: Tracking Net dataset attributes description. [40]

3.1.4 Object Tracking Benchmark (OTB)

Object Tracking Benchmark contains 50 or 100 fully labelled video sequences. The authors provide the perturbation in initialization to get more objective results. This dataset provides also a benchmark for evaluation. [41] Tags description is shown in Fig 3.3.

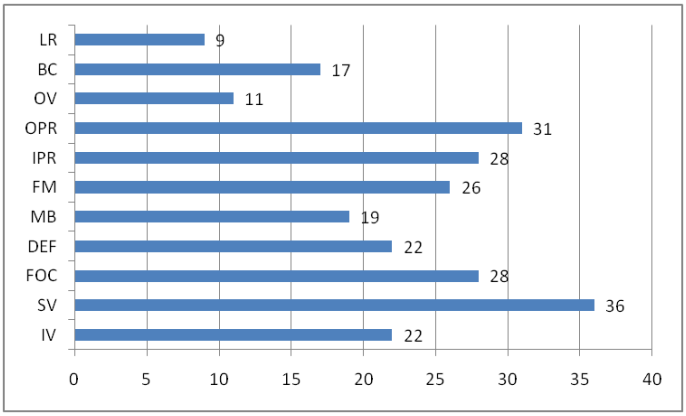

Fig. 3.3: TB-50 dataset attributes description.

3.1.5 ALOV300++

The Amsterdam Library of Ordinary Videos for tracking is a dataset that contains 314 videos. The average duration is 9.2 seconds. The aim is to cover a variety of scenarios. In total it consists of 89,964 labelled frames. Tags description is shown in Fig 3.4.

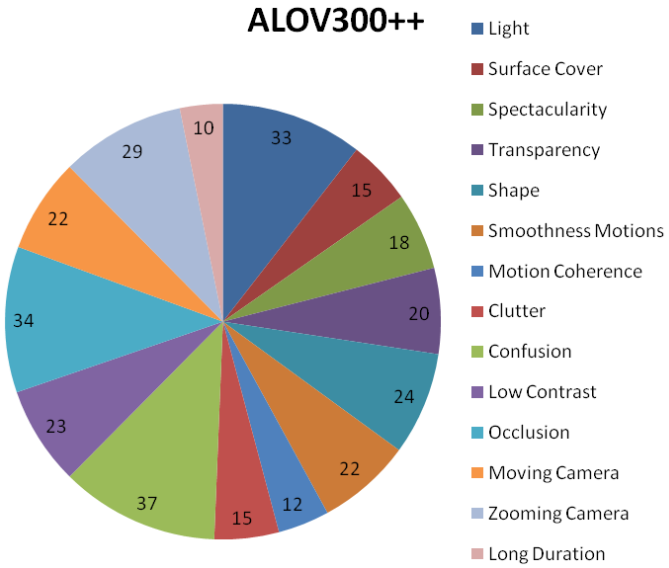

Fig. 3.4: ALOV dataset attributes description with all tags.

3.1.6 ImageNet

The ImageNet contains over 22 thousand categories and 15 millions images. The part Large scale visual recognition challenge contained 1000 categories and 1,431,167

images. State of the Art detecting algorithms were learned on this dataset. The focus is on object detection in an image or in a video. The dataset was developed over 7 years, from 2010 to 2017.

3.2 Performance Measurements

The evaluation is performed by several metrics. The metrics have to be objective. The base is to evaluate if the classification is correct. For this decision, we use the following marks:

- True Positive (TP) - is a detection example, which is correct.
- False Positive (FP) - is a detection example, which is not correct.
- False Negative (FN) - is a target example, which is not found.
- True Negative (TN) - is no response to no target.

An example of image classification is shown on Fig 3.5. The TN is tricky for image tasks because usually only several targets are marked and the rest is not marked, so there can be a huge number of TN. In different fields, for example in medicine, it is also an important value.

3.2.1 Precision-recall Curve, AP, mAP

This metric is more important for detectors, but in the future it may be used for tracking as well. From the detector, we want two abilities. The first one is *precision*, we want to eliminate the number of false-positive examples. The second is *recall*, we want to find all ground truth examples. Optimizing these two metrics is usually a trade-off. When we get high precision, we mark fewer objects. From the opposite side, when we want to find all ground truth targets, it is useful to find more objects.

The precision is defined in equation 3.1. The symbol # denotes the number of examples. The recall is defined in equation 3.2. Evaluation can be done on the precision-recall curve shown on Fig 3.6. The AP defined by equation 3.4 is the metric for evaluation. The AP follows the idea that both precision and recall should be high.

$$Precision(c) = \frac{\#TP(c)}{\#TP(c) + \#FP(c)} \tag{3.1}$$

$$Recall(c) = \frac{\#TP(c)}{\#TP(c) + \#FN(c)} \tag{3.2}$$

$$AP = \frac{1}{11} \sum_{r \in (0,0.1,0.2,\ldots,1)} p_{interp(r)} \tag{3.3}$$

$$p_{interp(r)} = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{3.4}$$

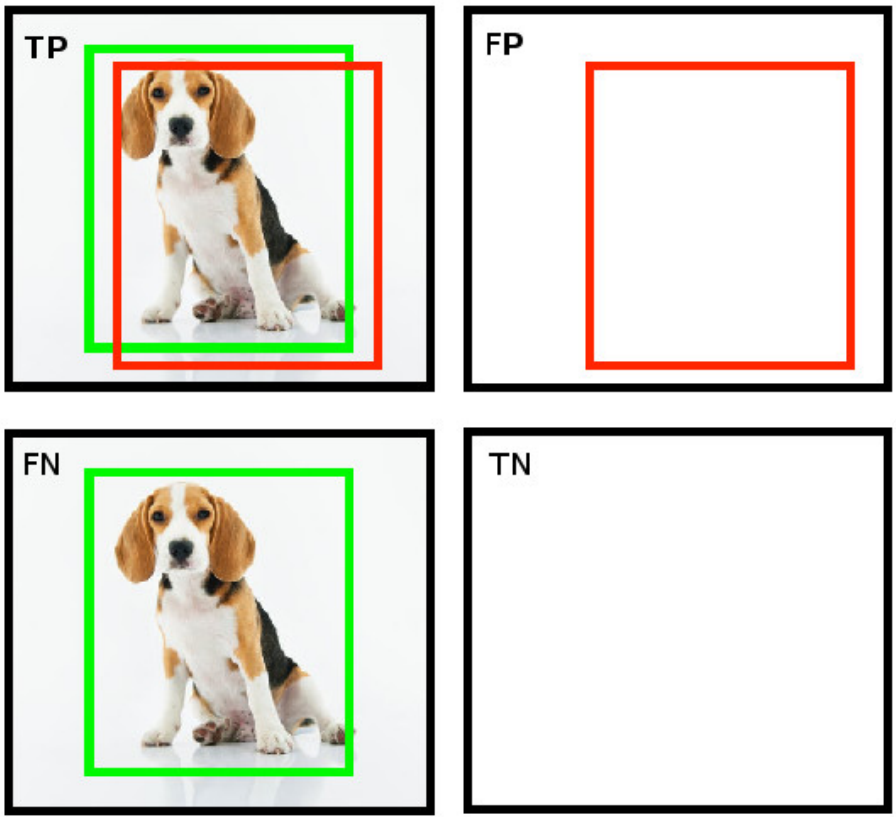

Fig. 3.5: TP,FP,FN,TN example, the mark is in the left corner: red is estimate bounding box, green is groundtruth bounding box [45]

If we have more classes for our task, we can compute mAP, defined in equation 3.5.

$$mAP = \frac{1}{\#classes} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)} \tag{3.5}$$

3.2.2 Center Error

The simplest method evaluating the displacement is the center error. The target is represented by one point. The error is computed by a metric over a distance of two points. The distance used the root-mean-square-error:

$$RMSE(\Lambda^G, \Lambda^T) = \sqrt{\frac{1}{N} \sum_{t=1}^{N} ||x_t^G - x_t^T||^2} \tag{3.6}$$

where $\Lambda^G = (R_t^G, x_t^G)_{t=1}^N$ is a set of all ground truth bounding boxes described by center position x and region of the object R. The same is true for the target set denoted Λ^T. The advantage of this metric is its simplicity. The drawback of this method is that it is sensitive to the subjective annotation of the center point. [42]

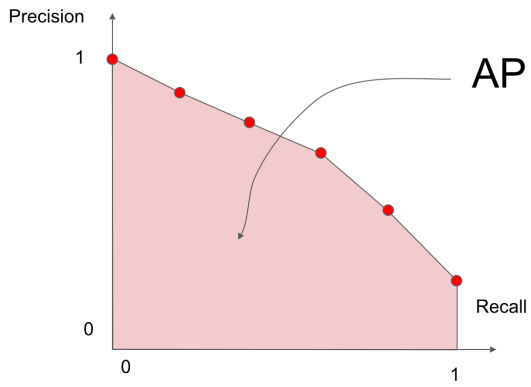

Fig. 3.6: example of the Average Precision on precision-recall curve. [45]

3.2.3 Intersection over Union

This metric is used for classifying an object region with a ground truth region, i.e. computing succession. The IoU is the ratio of the intersection over the union. It determines how well the estimated bounding box corresponds with the ground truth bounding box. Commonly, an IoU >= 0.5 means a hit, otherwise, it is a fail. If we want higher restriction, i.e. to achieve better performance (up to 1, the ideal), we can denote this with mAP@p, where $p \in (0, 1)$ is the IoU. Common other thresholds are 0.75 and 0.95.

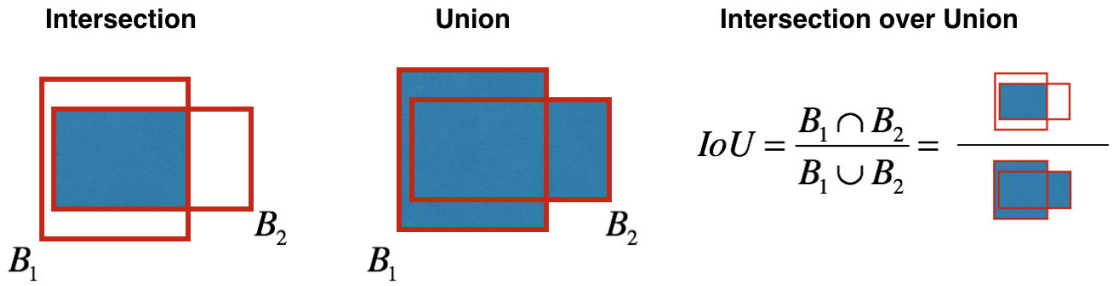

Fig. 3.7: Intersection over Union example. [47]

3.2.4 Tracking Accuracy vs. Robustness

The tracking accuracy is understood as the IoU metric. The robustness is counted from the number of tracker failures, called *failure rate*. The A-R metric is defined:

$$A - R(\Lambda^G, \Lambda^T) = (\Phi(\Lambda^G, \Lambda^T), F_0(\Lambda^G, \Lambda^T)) \tag{3.7}$$

where Φ is the *average overlap* and F_0 denotes the failure rate.

3.3 Embedded Devices

In this section, some embedded devices capable to perform image processing will be presented. In our context, the embedded device is a small single-board computer.

3.3.1 Raspberry Pi

Raspberry Pi is the small single-board computer, whose dimensions are the same as a credit card. Behind this platform is a wide community. The price is in the tens of dollars. The foundation came from the United Kingdom. Thanks to its low price and large community, it is widely used in robotics. The CPU and GPU are in one package (Broadcom BCM2837) called System on a Chip. The power adapter is 5 V/2.5 A. Other specifications are shown in Tab. 3.1.

Tab. 3.1: Specifications RaspberryPI 3 B. Available from URL: <`https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/`>

CPU	Quadro core ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless
Display	HDMI HD, Composite
Flash	None
Dimensions	85.60 × 56.5 mm

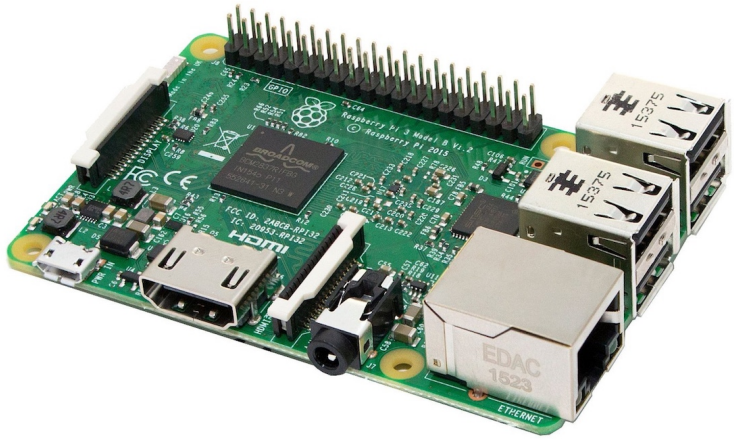

Fig. 3.8: RaspberryPI 3 model B [35]

3.3.2 ODROID

The ODROID tries to be an alternative for the Raspberry Pi. It has slightly better performance for a similar price, but it does not have the advantage of a large community, which the Raspberry Pi has. The platform is developed by the company Hardkernel from South Korea. Another advantage is a huge solid heat sink. The power adapter is 5 V/2.5 A. Other specifications are shown in Tab. 3.2.

Tab. 3.2: Specifications ODROID-C2. Available from URL: <`https://www.hardkernel.com/shop/odroid-c2/`>

CPU	Quadro core Cortex-A53, 1.5GHz
GPU	ARM Mali-450 MP 700MHz
RAM	2GB 32bit DDR3 912MHz
Networking	Gigabit Ethernet
Display	HDMI 2.0 4K/60Hz display
Flash	eMMC
Dimensions	85.6 × 56.5 mm

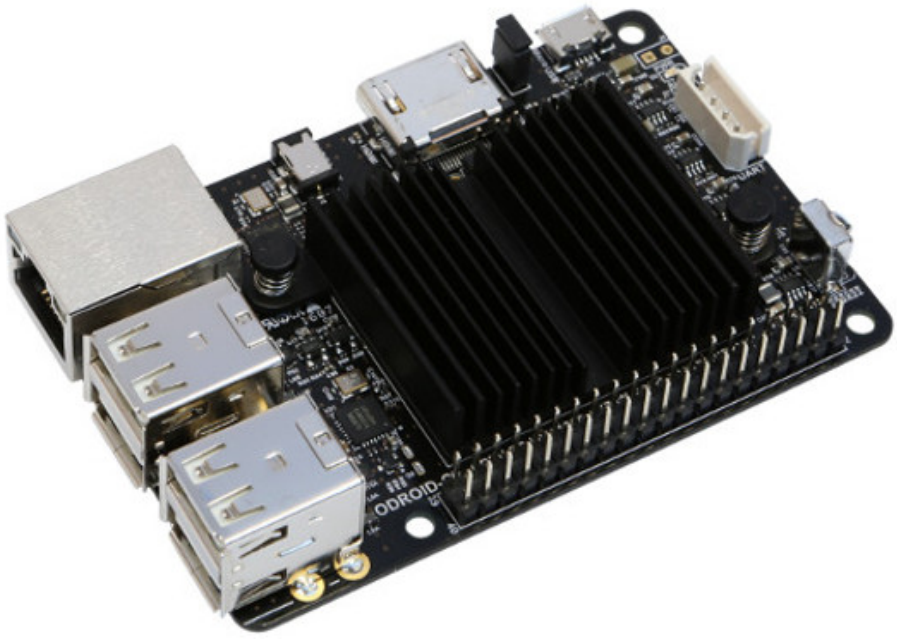

Fig. 3.9: ODROID-C2. Available from URL: <`https://www.hardkernel.com/shop/odroid-c2/`>

3.3.3 Nvidia Jetson

The device is developed by Nvidia, especially for artificial intelligence. The company developed more types TX, TK, nano. Specifications are shown in Tab. 3.3.

Tab. 3.3: Specifications Nvidia Jetson TX2 4GB. Available from URL: <https://developer.nvidia.com/embedded/develop/hardware>

CPU	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores
GPU	Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex
RAM	4 GB 128-bit LPDDR4
Networking	None
Display	HDMI 2.0, DP 1.2, eDP 1.4, Two 1x4 DSI
Flash	16GB eMMC 5.1
Dimensions	50.0 x 90.0 mm

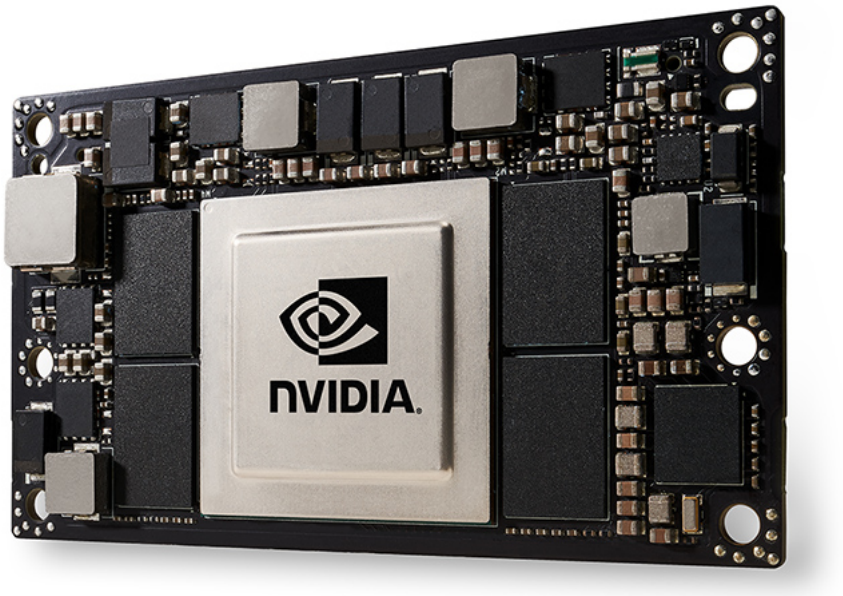

Fig. 3.10: Nvidia Jetson TX2 from nvidia sites. Available from URL:<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/>

3.3.4 Beagleboard

The platform is developed by Texas Industries. Specifications are shown in Tab. 3.4.

Tab. 3.4: Specifications BeagleBone Black [36]

CPU	Sitara AM3358BZCZ100 1GHz, 2000 MIPS
GPU	SGX530 3D, 20M Polygons/S
RAM	512MB DDR3L 800MHZ
Networking	10/100 Ethernet
Display	HDMI HD
Flash	4GB 8-bit eMMC
Dimensions	86,3 x 53,3 mm

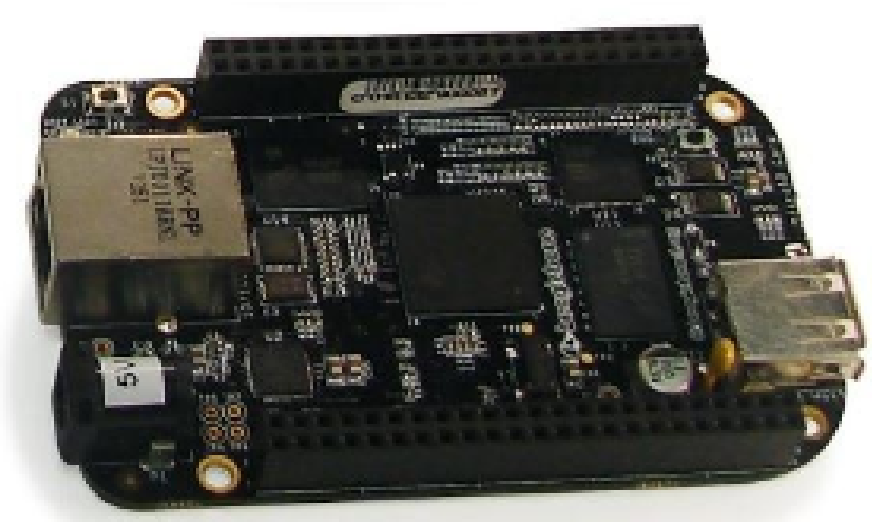

Fig. 3.11: BeagleBone Black [36]

4 Practical Part

The practical part consists of implementation in a mobile workstation, re-implementation on an embedded device, dataset creation, and evaluation. The work was done as follows: in the previous chapters, methods for implementation were found, datasets and metrics for evaluation were introduced, and embedded devices were presented.

Compared algorithms, chosen for experiments, are AOB, MIL, KCF, TLD, MOSSE, Median Flow, GOTURN and CSR-DCF due to their ability run in real time. For a device, Raspberry Pi was chosen due to its availability, good performance and wide community. Most of the trackers are short-term methods, that is why the VOT dataset was chosen for evaluation. The chosen version of VOT is VOT2014 with 25 videos and two base experiments: baseline and region noise.

4.1 Prerequisites

The primary step before the implementation is to consider dependencies. For implementation and reimplementation OS Linux is used, therefore, the dependencies are focused on this environment.

4.1.1 Caffe

For the original GOTURN implementation, it was necessary to compile and install the Deep Learning framework Caffe. The compilation has those dependencies: *CUDA* required for GPU mode. *Blas* via ATLAS, MKL or OpenBLAS, *Boost* >= 1.55, *protobuf, glog, gflags* and *hdf5*. The optional dependencies are *OpenCV* >= 2.4, input-output libraries *lmdb, leveldb* and *cuDNN* for GPU acceleration.

4.1.2 OpenCV

OpenCV is an open source computer vision library. This library congregates an implementation of diverse computer vision tasks, for example it provides modules for object detection, image processing, video analysis, or tracking.

The compilation on Linux has those dependencies: compiler *GCC* 4.4.x or later, *CMake* 2.8.7 or higher, *Git* repositories manager, *GTK+*2.x or higher include headers, *pkg-config*, *Python* 2.6 or later with *Numpy* 1.5 or later package with developer packages and *ffmpeg* or *libav* development packages. The optional dependencies are *libtbb2, libtbb-dev*, *libdc1394* 2.x, *libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev* and *CUDA Toolkit* 6.5 or higher.

4.1.3 CUDA

The interface between programs and a GPU is provided by the Compute Unified Device Architecture (CUDA). It is a framework from the company Nvidia for GPU support. For a neural network, CUDA has an extension called cuDNN. Raspberry Pi has its own GPU, but it is not supported by CUDA.

4.1.4 Matlab, Octave

Matlab is a mathematical and scientific environment. It is created with a matrix numbers representation. Its advantage is a large-scale community with lots of adds-in. Debugging in this environment is quite easy because all variables can be seen.

GNU Octave is a scientific environment similar to Matlab developed for Linux. The Octave Forge is a central location for development packages. The most of Matlab scripts are compatible with Octave and otherwise. For the VOT analysis, it was necessary to install the packages *io, image* and *statistics* from Octave Forge.

4.2 Implementation

For the implementation, a mobile workstation (Elitebook 8770w) was chosen, with *eight Intel Core i7-3740QM CPU 2,70GHz* processors and a *Nvidia Quadro K3000M* graphics card. The operating system is Ubuntu 16.04 LTS.

For the implementation, the open library OpenCV, version 3.4.1, was used. Version 4.1.0 were also tried. In terms of the effort to achieve the best performance, however, the version 4.1.0 ran slower than the version 3.4.1, therefore, the version 3.4.1 was chosen.

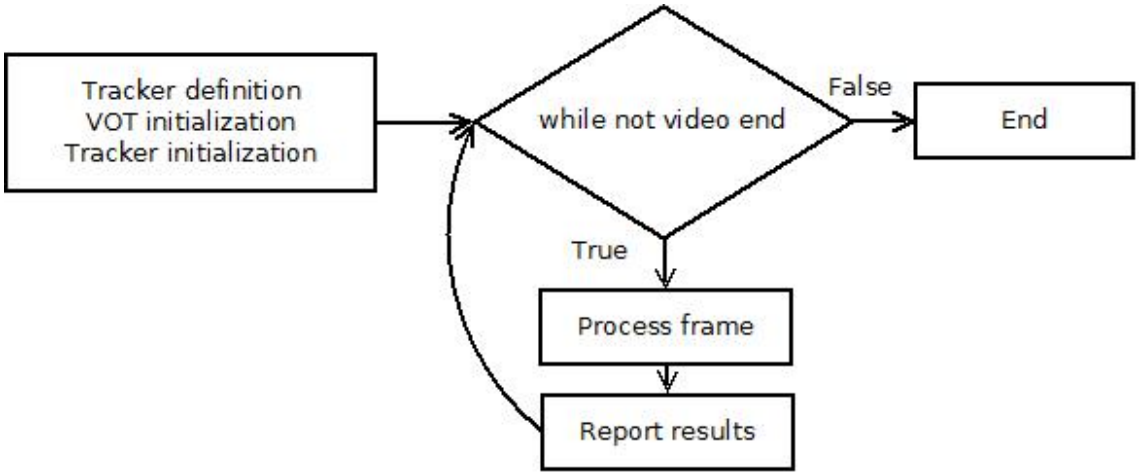

Fig. 4.1: VOT connection program block diagram

It was necessary to write a program to connect the tracker and the datasets. The program was inspired by examples from the VOT dataset. The programming language c++ was chosen for the implementation. The program is constructed as follows: First, it creates the tracker and the VOT object. Then, it reads the first frame and initializes the tracker. Then, it goes to the loop, where it reads all frames from video, estimates new target position and reports the result. The schema is shown on Fig 4.1. The program is attached in the appendix B.

Another program for testing the implementation was implemented and it is called the Tracker. This program was inspired by learning OpenCV site [43]. The program has two main branch. One is branch for reading the VOT sequences of video frames. The second branch reads frames from the video. It was necessary to create *CMakeList.txt* for compilation with CMake. It was written bash script *run.sh* for running the Tracker on Linux. Both are included on the CD.

4.3 Reimplementation

Raspberry Pi 3 was chosen as our embedded device. The specification can be seen in section 3.3.1. The reason for this choice was availability and good performance. The OS was Raspbian, specifically built for Raspberry Pi. It is an OS based on Debian and has plenty of tools built for the Raspberry Pi.

One important point is the chip temperature. For the evaluation, it was necessary to add a heat sink, which is only optional for Raspberry Pi, and also add active cooling with a ventilator. When we use all computing power, the temperature can rise to 80 °C without these components. The temperature rise in time of stress test is shown on Fig 4.2. The stress test tests the device stability work under load.

With the chip temperature, is close connect the overclocking. This term is used to increase the clocking frequency of CPU. By this operation, we can achieve a significant boost in power. But it is connected to a higher temperature and power consumption. Therefore, it is necessary to have active cooling on Raspberry Pi. The overclocking on Raspberry Pi is simple because Raspberry Pi provides a tool for it.

4.4 Achieved Results

For results, VOT 2014 dataset containing 25 videos was used. Every video was tested in two experiments on both devices. The experiments were baseline and region-noise. The number of all trackers is nine because eight trackers were implemented and, for comparison, also the original GOTURN implementation was used. But the original

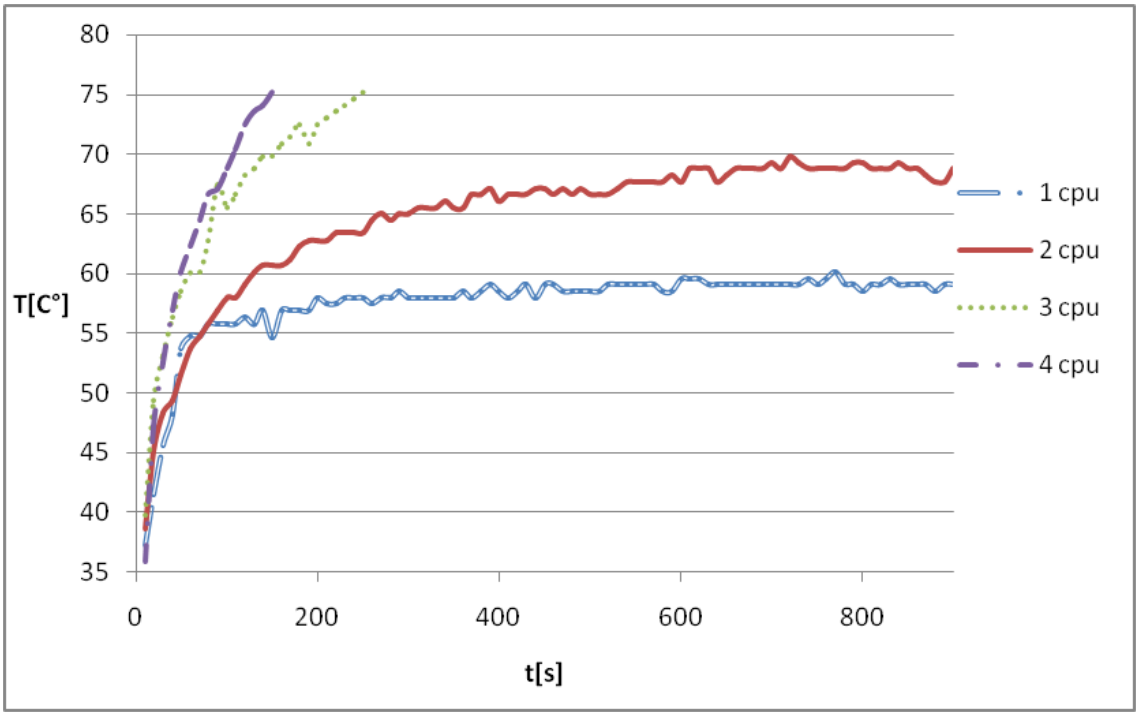

Fig. 4.2: Temperature rise in time of stress test on Raspberry Pi with heat sink without active cooling. It is necessary to note that the experiment was interrupted when the temperature reaches 75 C°

implementation needs the CUDA, thus only for mobile workstation, the comparison was made.

The first part is an evaluation on a mobile workstation. On Fig 4.3 are shown A-R rank, where the robustness is pooled and transformed into range 0,1. The best A-R rank has the CSR-DCF tracker but due to his speed results, it cannot be used in real time. The speed is interesting, which is presented in the appendix C. Even on the mobile workstation, not all algorithms can run real time. Another interesting things is that original implementation of GOTURN has significantly better results. This may be caused used model, or it can be due to implementation. The raw results are in an appendix C.

From the result for Raspberry Pi, it is observed that only two algorithms are able to run real-time on the chosen embedded device: Median Flow and MOSSE. The best candidate is Median Flow, despite lower FPS than the MOSSE, it has a lower number of failure. Interesting can also be the KCF tracker with an average 15 FPS, but it has a too high number of failures. Detailed results are in an appendix D. Detailed accuracy and robustness results are on included CD.

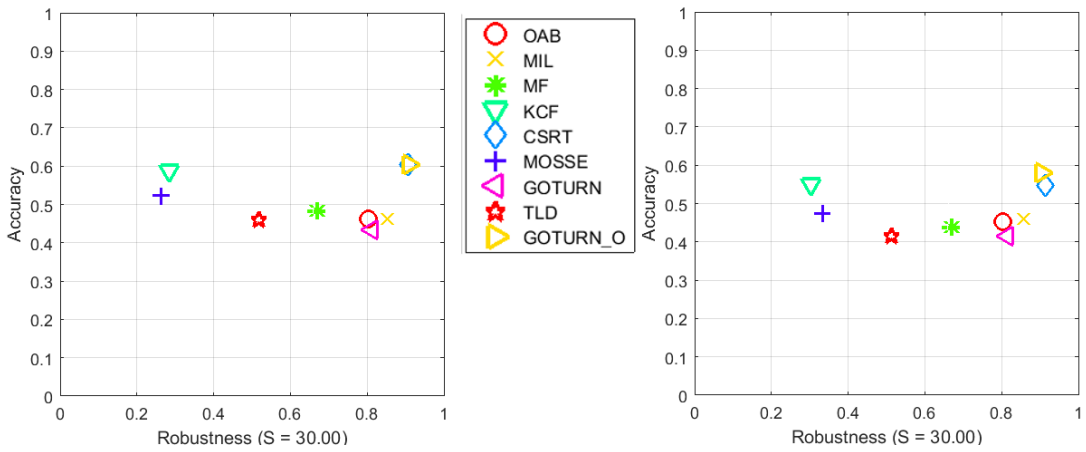

Fig. 4.3: A-R rank for algorithms on mobile workstation: Left is the baseline experiment and right is region_noise experiment.

4.5 Creating a Dataset

Learning algorithms need a *training* dataset to shape themselves to the problem. In our case, it is a dataset, which consists of video sequences. The training process has one weakness: a trained model can overfit on the training dataset. One practice how to prevent overfitting is creating a *validation* dataset. The process works as follows: first, start to train a model on the training dataset. During the training, compute the output for a validation dataset, in addition to the training dataset. The loss should still descend for both datasets. If the loss for the validation dataset starts to rise, then the model is starting overfit the training dataset. This problem is shown in Fig 4.5. The validation dataset can be sampled from the original training dataset, i.e. it can be a part of the original training dataset. The important fact is that the model is not trained on a validation dataset, i.e. it simulates test dataset.

The training dataset must be as diverse as much as possible. The goal is to construct a dataset for learning a general object tracker, e.g. GOTURN tracker. It is common that in the dataset most of the videos are with people target. Suggested dataset consists of 30 videos from multiple datasets. The content is as follows: from OTB50: *Bird1, Box, Car1, David, Deer, Panda, RedTeam, MotorRolling*, from the ALOV300++: *Light5, Light15, SurfaceCover3, SurfaceCover5, SurfaceCover9, Spectacularity2, Spectacularity8, Transparency1, Transparency5, Transparency8, Transparency14, Shape3, Shape11, Shape17, Shape22, MotionCoherence4 and Clutter6*, and from VOT2014: *ball, fernando, fish1, hand2, and polarbear.*

As was mentioned a validation datasets can be sampled from the training dataset. Good results are obtained with a random sampling of examples from the training

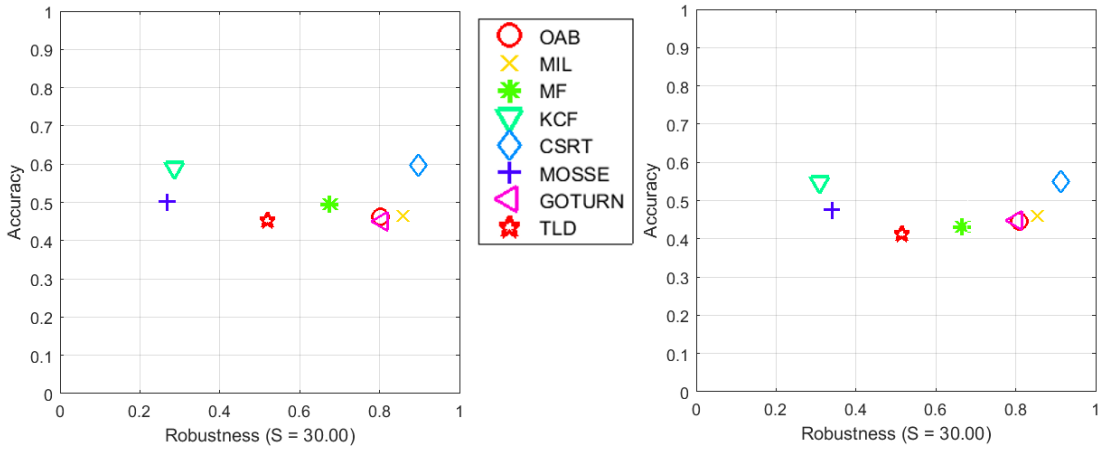

Fig. 4.4: A-R rank for algorithms on Raspberry Pi: Left is the baseline experiment and right is region_noise experiment.

dataset and ratio 8:2, which means that the training dataset will have 80% of the original data and a validation dataset will have 20% of data. Once the data are split, the training can starts.

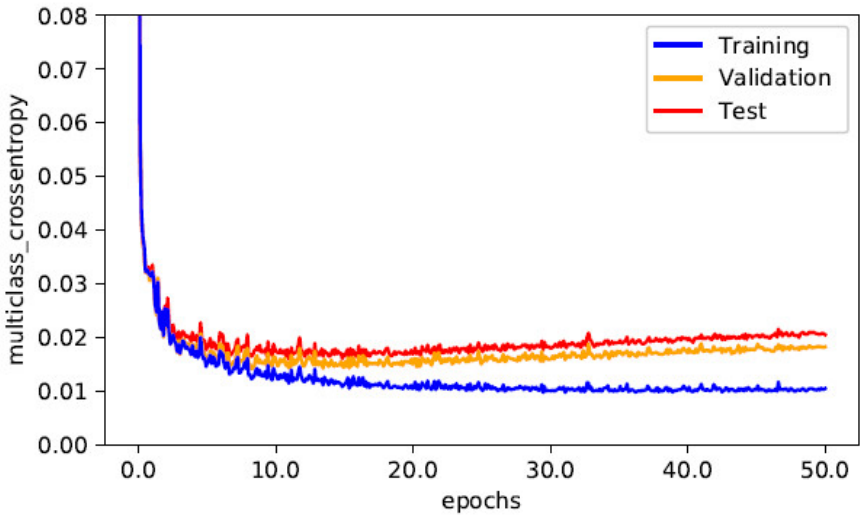

Fig. 4.5: Example of the overfitting on the training dataset. [46]

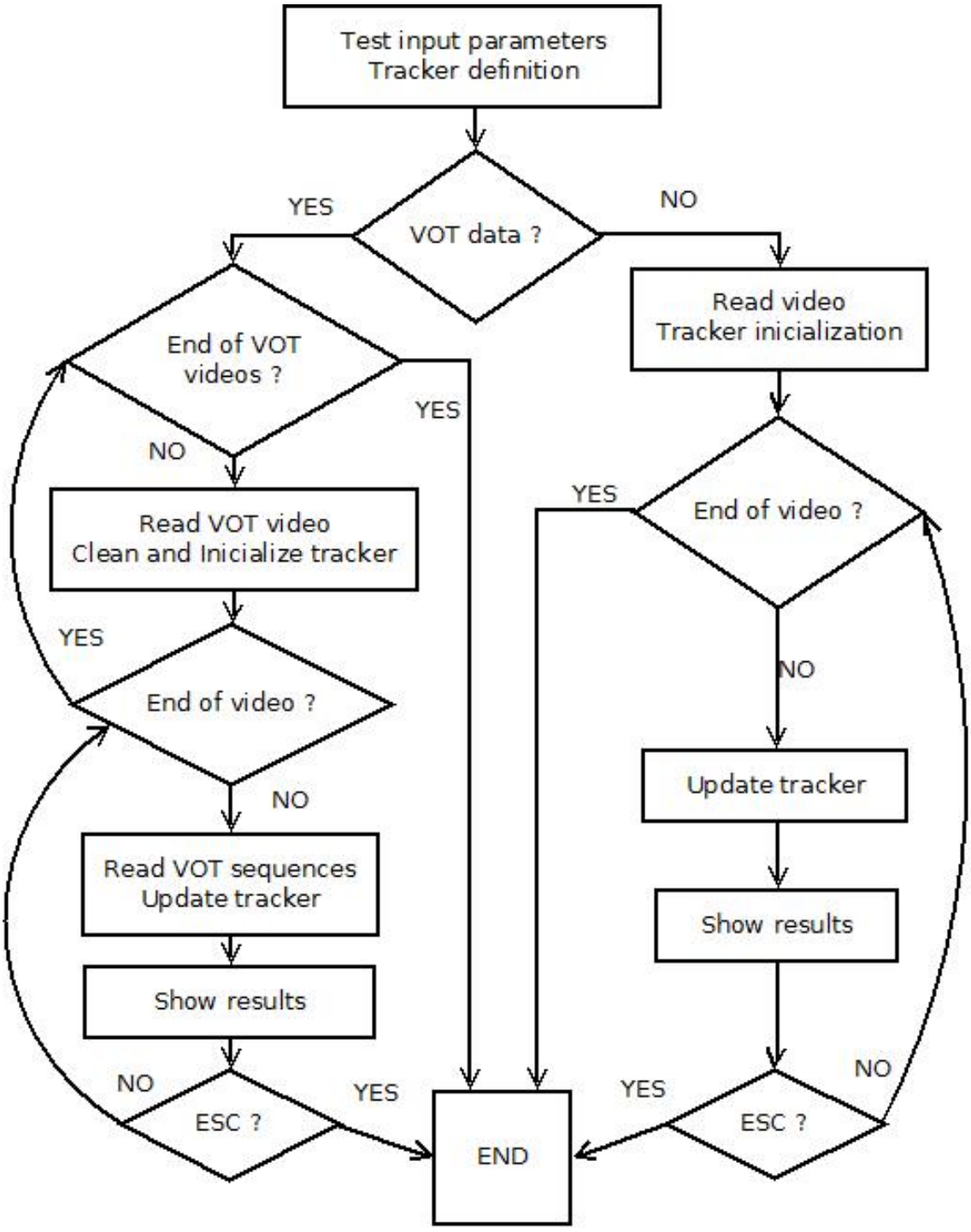

Fig. 4.6: The Tracker block diagram

5 Conclusion

This thesis is focused on algorithms for object tracking. Especially, on robust algorithms, which can work in general (unknown) scene. This problematic is widely studied from the inception of the computer vision. Many algorithms were published. Some of them use simple boosting methods, others use the correlations filters or neural networks.

Firstly, some theoretical background were introduced in the first chapter. The chapter summarizes basics from computer vision field, the main focus is on the image features. The next chapter presented some constraints and assumptions used for object tracking. Different approaches used for different cases were discussed. Then several algorithms were presented. The third part was focused on the evaluation. Datasets and metrics were named and described. In this chapter also the embedded devices were presented.

Based on the received knowledge were chosen algorithms AOB, MIL, KCF, TLD, MOSSE, Median Flow, GOTURN and CSR-DCF, for implementation, due to their ability run in real time. Algorithms were successfully implemented and then reimplement in the chosen embedded device. The chosen embedded device was Raspberry Pi because of the community and its availability.

Implemented algorithms were evaluated on VOT2014 dataset. From the results, it was found out that only two algorithms are able to run real time on chosen embedded device. In addition, two different implementation of algorithm were evaluated on a mobile workstation. It was found out that the original implementation reached better results. Also a new training dataset was constructed and it was proposed a method how to sample the validation dataset from the training dataset.

The future focus of this thesis can be using overclocking for boosting the performance. Also adding the video grabber can be usefull. Interesting can also be a comparison of different embedded devices. Many different tasks can be deal with this program, for example remote control using the camera, or simple security system. This thesis can serve as a brief introduction into the general object tracking problematic.

Bibliography

[1] YILMAZ, A.; JAVED, O.; SHAH, M. *Object tracking: A survey. ACM Computing Surveys (CSUR).* ACM, 2006, 38(4), 13-es [cit. 2019.01.12]. DOI: 10.1145/1177352.1177355.

[2] PAREKH, Himani S.; THAKORE, Darshak G.; JALIYA, Udesang K. *A Survey on Object Detection and Tracking Methods*International Journal of Innovative Research in Computer and Communication Engineering, 2014, 2.2: 2970-2978. Available from URL: <https://pdfs.semanticscholar.org/25a6/c5dff9a7019475daa81cd5a7f1f2dcdb5cf1.pdf>.

[3] ATHANESIOUS, J. Joshan; SURESH, P. *Systematic survey on object tracking methods in video* International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 2012, 1.8: pp: 242-247.

[4] TRAJCEVSKI, Goce; SCHEUERMANN, Peter. *Targets and Shapes Tracking (Advanced Seminar)* In: 2018 19th IEEE International Conference on Mobile Data Management (MDM). IEEE, 2018. p. 7-10. Available from URL: <https://ieeexplore.ieee.org/abstract/document/8411255>.

[5] HORÁK, K. *Learning materials to subject computer vision, Lecture 11.* [online], [cit. 24.1.2019] Available from URL: <http://midas.uamt.feec.vutbr.cz/POV/pov_cz.php>.

[6] SALARI, Vali; SETHI, Ishwar K. *Feature point correspondence in the presence of occlusion.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1990, 12.1: 87-91.

[7] MUHAMMAD, Rashid Bin. *Greedy Introduction.* [online], [cit. 24.1.2019]. Available from URL: <http://www.personal.kent.edu/%7Ermuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm>.

[8] VEENMAN, Cor J.; REINDERS, Marcel JT; BACKER, Eric. *Resolving motion correspondence for densely moving points.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 1: 54-72.

[9] BROIDA, Ted J.; CHELLAPPA, Rama. *Estimation of object motion parameters from noisy images.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, 1: 90-99.

[10] BAR, Shalom Y.; FORTMANN, T. E. *Tracking and data association.* 1988. PhD Thesis. Academic Press.

[11] STREIT, Roy L.; LUGINBUHL, Tod E. *Maximum likelihood method for probabilistic multihypothesis tracking.* In: Signal and Data Processing of Small Targets 1994. International Society for Optics and Photonics, 1994. p. 394-406.

[12] COMANICIU, Dorin; RAMESH, Visvanathan; MEER, Peter. *Kernel-based object tracking.* IEEE Transactions on pattern analysis and machine intelligence, 2003, 25.5: 564-577.

[13] SHI, Jianbo; TOMASI, Carlo. *Good features to track.* Cornell University, 1993.

[14] POLIKAR, Robi. *Ensemble learning* [online] [cit. 30.7.2019]. Available from URL:
<http://scholarpedia.org/article/Ensemble_learning>.

[15] TAO, Hai; SAWHNEY, Harpreet S.; KUMAR, Rakesh. *Object tracking with bayesian estimation of dynamic layer representations.* IEEE transactions on pattern analysis and machine intelligence, 2002, 24.1: 75-89.

[16] BLACK, Michael J.; JEPSON, Allan D. *Eigentracking: Robust matching and tracking of articulated objects using a view-based representation.* International Journal of Computer Vision, 1998, 26.1: 63-84.

[17] AVIDAN, Shai. *Support vector tracking.* IEEE transactions on pattern analysis and machine intelligence, 2004, 26.8: 1064-1072.

[18] HUTTENLOCHER, Daniel P.; NOH, Jae J.; RUCKLIDGE, William J. *Tracking non-rigid objects in complex scenes.* In: 1993 (4th) International Conference on Computer Vision. IEEE, 1993. p. 93-101.

[19] SATO, Koichi; AGGARWAL, Jake K. *Temporal spatio-velocity transform and its application to tracking and interaction.* Computer Vision and Image Understanding, 2004, 96.2: 100-128.

[20] KANG, Jinman; COHEN, Isaac; MEDIONI, Gerard. *Object reacquisition using invariant appearance model.* In: Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on. IEEE, 2004. p. 759-762.

[21] POPPE, R. *Condensation-conditional density propagation for visual tracking.* Computer Vision Image Understanding, 2007, 108: 4-18.

[22] BERTALMIO, Marcelo; SAPIRO, Guillermo; RANDALL, Gregory. *Morphing active contours.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22.7: 733-737.

[23] RONFARD, Rémi. *Region-based strategies for active contour models.* International journal of computer vision, 1994, 13.2: 229-251.

[24] SOBKOVÁ, Eva. *Sekvenεční Monte Carlo metody.* Praha, 2012. Bachelor thesis. Univerzita Karlova v Praze. Fakulta Matematicko-fyzikální. Available from URL:
<https://is.cuni.cz/webapps/zzp/detail/115669/>.

[25] BABENKO, Boris; YANG, Ming-Hsuan; BELONGIE, Serge. *Visual tracking with online multiple instance learning.*CVPR,2009,[cit. 15.7.2019]. Available from URL:
<http://vision.ucsd.edu/~bbabenko/data/miltrack_cvpr09.pdf>.

[26] HELD, David; THRUN, Sebastian; SAVARESE, Silvio. *Learning to Track at 100 FPS with DeepRegression Networks.* 2016, European Conference Computer Vision (ECCV).[cit. 15.7.2019] Available from URL:
<http://davheld.github.io/GOTURN/GOTURN.pdf>.

[27] HENRIQUES, João F.; CASEIRO, Rui; MARTINS, Pedro; BATISTA, Jorge. *High-Speed Tracking with Kernelized Correlation Filters* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015,[cit. 15.7.2019]. Available from URL:
<https://arxiv.org/pdf/1404.7584.pdf>.

[28] KALAL, Zdenek; MIKOLAJCZYK, Krystian; MATAS, Jiri. *Forward-backward error: Automatic detection of tracking failures.* 20th International Conference on Pattern Recognition,2010,[cit. 15.7.2019]. Available from URL:
<https://dspace.cvut.cz/bitstream/handle/10467/9553/2010-forward-backward-error-automatic-detection-of-tracking-failures.pdf?sequence=1&isAllowed=y>.

[29] KALAL, Zdenek; MIKOLAJCZYK, Krystian; MATAS, Jiri. *Tracking-Learning-Detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence,2012,[cit. 15.7.2019]. Available from URL:
<http://kahlan.eps.surrey.ac.uk/featurespace/tld/Publications/2011_tpami>.

[30] GRABNER, Helmut; GRABNER, Michael; BISCHOF, Horst. *Real-time tracking via on-line boosting.* Proceedings British Machine Vision Conference, 2006, [cit. 15.7.2019]. Available from URL:
<http://www.macs.hw.ac.uk/bmvc2006/papers/033.pdf>.

[31] LUKEŽIČ, Alan; VOJÍŘ, Tomáš; ZAJC, Luka Čehovin; MATAS Jiří; KRISTAN Matej. *Discriminative Correlation Filter Tracker with Channel and Spatial Reliability.* International Journal of Computer Vision, 2018, [cit. 15.7.2019]. Available from URL: <https://arxiv.org/pdf/1611.08461.pdf>.

[32] FREUND, Yoav; SHAPIRE, Robert. *A Decision-Theoretic Generalization of On-Line Learningand an Application to Boosting.* Journal of computer and system sciences, 1996, [cit. 26.7.2019]. Available from URL: <http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf>.

[33] BOLME, David S.; BEVERIDGE, J. Ross; DRAPER, Bruce A.; LUI, Yui Man. *Visual Object Tracking using Adaptive Correlation Filters.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, [cit. 26.7.2019]. Available from URL: <https://www.cs.colostate.edu/~vision/publications/bolme_cvpr10.pdf>.

[34] DALAL, Navneet; TRIGGS, Bill. *Histograms of Oriented Gradients for Human Detection.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, [cit. 25.7.2019]. Available from URL: <https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>.

[35] BRUNNER, Grant. *ET deals: Get a Raspberry Pi 3 Model B on Amazon.*[online][cit. 19.7.2019]. Available from URL: <https://www.extremetech.com/deals/231905-et-deals-save-on-raspberry-pi-3>.

[36] KRIDNER, Jason; COLEY, Gerald. *System Reference Manual*[online][cit. 19.7.2019]. Available from URL: <https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual>.

[37] LUCAS, Bruce David. *Generalized Image Matching by the Method of Differences.* 1984, PhD Thesis. Available from URL: <https://www.ri.cmu.edu/pub_files/pub4/lucas_bruce_d_1984_1/lucas_bruce_d_1984_1.pdf>.

[38] LEAL-TAIXÉ, Laura; MILAN, Anton; REID, Ian; ROTH, Stefan and SCHINDLER, Konrad. *MOTChallenge 2015:Towards a Benchmark for Multi-Target Tracking*, 2015,[cit. 25.7.2019]. Available from URL: <https://arxiv.org/pdf/1504.01942.pdf>.

[39] KRISTAN, Matej; MATAS, Jiri; LEONARDIS, Aleš; at all. *A Novel Performance Evaluation Methodology for Single-Target Trackers* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016, [cit. 26.7.2019]. Available from URL: <https://arxiv.org/pdf/1503.01313.pdf>.

[40] MÜLLER, Matthias; BILI, Adel; GIANCOLA, Silvio; at all. *TrackingNet: A Large-Scale Dataset and Benchmark for Object Tracking in the Wild* European Conference on Computer Vision, 2018, [cit. 30.7.2019]. Available from URL: <https://arxiv.org/pdf/1803.10794v1.pdf>.

[41] WU, Yi; LIM, Jongwoo; YANG, Ming-Hsuan. *Online Object Tracking: A Benchmark* IEEE Conference on Computer Vision and Pattern Recognition, 2013, ISBN: 978-1-5386-5672-3 [cit. 30.7.2019]. Available from URL: <http://faculty.ucmerced.edu/mhyang/papers/cvpr13_benchmark.pdf>.

[42] ČEHOVIN, Luka; LEINARDIS, Aleš; and KRISTAN, Matej. *Visual object tracking performance measures revisited* IEEE Transactions on Image Processing, 2016, [cit. 30.7.2019]. Available from URL: <https://arxiv.org/pdf/1502.05803.pdf>.

[43] MALLICK, Satya. *Object Tracking using OpenCV (C++/Python)*[online][cit. 30.7.2019]. Available from URL: <https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>.

[44] LINDSETH, Frank. *Learning materials to subject TDT4265 Computer Vision and Deep Learning: Object detection*Spring 2019

[45] NGUYEN, Hai Thanh. *Learning materials to subject TDT4173 Machine Learning and Case-based Reasoning, Lecture 10.*Spring 2019

[46] WINTEL, Florian; KRATOCHVILA, Lukáš. *Semester work to TDT4173 Machine Learning and Case-based Reasoning.*

[47] WINTEL, Florian; KRATOCHVILA, Lukáš. *Semester work to TDT4265 Computer Vision and Deep Learning.*

List of symbols, physical constants and abbreviations

SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
MGE	Modified Greedy Exchange algorithm
GOA	Greedy Optimal Assignment
JPDAF	Joint Probability Data Association Filtering
PMHT	Probabilistic Multiple Hypothesis Tracking
KLT	Kanade Lucas Tomasi feature tracker
SVM	Support Vector Machine
MHT	Multiple Hypothesis Tracking
HOG	Histogram of Oriented Gradients
RoI	Region of Interest
OAB	On-line Adaboost
KCF	Kernelized Correlation Filters
TLD	Tracking, Learning, and Detection
MIL	Multiple Instance Learning
MOSSE	Maximum Output Sum of Square Error
PSR	Peak to Sidelobe Ratio
GOTURN	Generic Object Tracking Using Regression Networks
CSR-DCF	Discriminative Correlation Filter with Channel and Spatial Reliability
VOT	Visual Object Tracking
MOT	Multiple Object Tracking
OTB	Object Tracking Benchmark
CNN	Convolutional Neural Network
IoU	Intersection over Union
mAP	mean Average Precision
AP	Average Precision
#	number of examples
SotA	State of the Art
FPS	Frames Per Second
OS	Operating System
CUDA	Compute Unified Device Architecture
CPU	Central Processor Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory

eMMC	embedded Multi-Media Controller
HDMI	High-Definition Multimedia Interface
OpenCV	Open source Computer Vision library
CNTK	The Microsoft Cognitive Toolkit
SV	Scale Variation
ARC	Aspect Ratio Change
FM	Fast Motion
LR	Low Resolution
OV	Out-of-View
IV	Illumination Variation
CM	Camera Motion
MB	Motion Blur
BC	Background Clutter
SOB	Similar Object
DEF	Deformation
IPR	In-Plane-Rotation
OPR	Out-of-Plane Rotation
POC	Partial Occlusion
FOC	Full Occlusion

List of appendices

A Tracker implementation

Listing A.1: The tracker implementation in language C++.

```
#include <opencv2/opencv.hpp>
#include <opencv2/tracking/tracking.hpp>
#include <opencv2/core/ocl.hpp>

#include "loader/loader_vot.h"
#include "helper/bounding_box.h"

using namespace std;
using namespace cv;

#define SSTR( x ) static_cast< std::ostringstream & >( \
( std::ostringstream() << std::dec << x ) ).str()

int main(int argc, char **argv)
{
    bool vot_dat;
    if (argc < 3) {
    std::cerr << "Usage: " << argv[0]
              << " number(BOOSTING,MIL,KCF,TLD,\
                  MEDIANFLOW,GOTURN,MOSSE,CSRT) video "
              << std::endl;
    return 1;
    }
    if (argc >= 4)
        vot_dat = false;
    else
        vot_dat = true;

    const string& videos_folder = argv[2];
    string trackerTypes[8] = {"BOOSTING", "MIL", "KCF", \
         "TLD", "MEDIANFLOW", "GOTURN", "MOSSE", "CSRT"};

    string trackerType = trackerTypes[atoi(argv[1])];
    Ptr<Tracker> tracker;

    printf("Tracker: %s\n",trackerType.c_str());

    if (trackerType == "BOOSTING")
```

```
39          tracker = TrackerBoosting::create();
40      if (trackerType == "MIL")
41          tracker = TrackerMIL::create();
42      if (trackerType == "KCF")
43          tracker = TrackerKCF::create();
44      if (trackerType == "TLD")
45          tracker = TrackerTLD::create();
46      if (trackerType == "MEDIANFLOW")
47          tracker = TrackerMedianFlow::create();
48      if (trackerType == "GOTURN")
49          tracker = TrackerGOTURN::create();
50      if (trackerType == "MOSSE")
51          tracker = TrackerMOSSE::create();
52      if (trackerType == "CSRT")
53          tracker = TrackerCSRT::create();
54
55      if (vot_dat){ // read a VOT sequences
56          LoaderVOT loader(videos_folder);
57          std::vector<Video> videos = loader.get_videos();
58
59          for (size_t video_num = 0; video_num < videos.size();
60             ++video_num) {
61              const Video& video = videos[video_num];
62              printf("Video: %zu\n", video_num);
63
64              int first_frame;
65              Mat frame;
66              BoundingBox bbox_gt;
67              video.LoadFirstAnnotation(&first_frame, &frame,
68                                        &bbox_gt);
69
70              Rect2d bbox(int(bbox_gt.x1_),int(bbox_gt.y1_),
71                          int(bbox_gt.get_width()),
72                          int(bbox_gt.get_height()));
73
74              tracker->clear();
75
76              if (trackerType == "BOOSTING")
77                tracker = TrackerBoosting::create();
78              if (trackerType == "MIL")
79                tracker = TrackerMIL::create();
```

```
80              if (trackerType == "KCF")
81                tracker = TrackerKCF::create();
82              if (trackerType == "TLD")
83                tracker = TrackerTLD::create();
84              if (trackerType == "MEDIANFLOW")
85                tracker = TrackerMedianFlow::create();
86              if (trackerType == "GOTURN")
87                tracker = TrackerGOTURN::create();
88              if (trackerType == "MOSSE")
89                tracker = TrackerMOSSE::create();
90              if (trackerType == "CSRT")
91                tracker = TrackerCSRT::create();
92
93              tracker->init(frame, bbox);
94
95              printf("Frames: ");
96              size_t frame_num = first_frame + 1;
97              for (; frame_num < video.all_frames.size();
98                 ++frame_num) {
99                imshow("Tracking", frame);
100               const bool draw_bounding_box = false;
101               const bool load_only_annotation = false;
102               bool has_annotation =
103                         video.LoadFrame(frame_num,
104                                         draw_bounding_box,
105                                         load_only_annotation,
106                                         &frame, &bbox_gt);
107               bbox_gt.Draw(0, 255, 0, &frame);
108               double timer = (double)getTickCount();
109               bool ok = tracker->update(frame, bbox);
110               float fps = getTickFrequency() /
111                       ((double)getTickCount() - timer);
112               if (ok)
113                   rectangle(frame, bbox,
114                         Scalar( 255, 0, 0 ), 2, 1 );
115               else
116                  putText(frame, "Tracking failure detected",
117                       Point(100,80),
118                       FONT_HERSHEY_SIMPLEX, 0.75,
119                       Scalar(0,0,255),2);
120
```

```
121                 putText(frame, trackerType + " Tracker",
122                     Point(100,20),
123                     FONT_HERSHEY_SIMPLEX, 0.75,
124                     Scalar(50,170,50),2);
125                 putText(frame, "FPS : " + SSTR(int(fps)),
126                     Point(100,50),
127                     FONT_HERSHEY_SIMPLEX, 0.75,
128                     Scalar(50,170,50), 2);
129                 imshow("Tracking", frame);
130                 // Exit if ESC pressed.
131                 int k = waitKey(1);
132                 if(k == 27) break;
133             } // end for
134         } // end for
135     } // end if
136     else{ // read a video file
137         VideoCapture video(argv[2]);
138 
139         if(!video.isOpened()){
140             cout << "Could not read video file" << endl;
141             return 1;
142         } // end if
143         Mat frame;
144         bool ok = video.read(frame);
145         Rect2d bbox = selectROI(frame, false);
146 
147         rectangle(frame, bbox, Scalar( 255, 0, 0 ), 2, 1 );
148 
149         tracker->init(frame, bbox);
150 
151         while(video.read(frame)){
152 
153             double timer = (double)getTickCount();
154             bool ok = tracker->update(frame, bbox);
155             float fps = getTickFrequency() /
156                   ((double)getTickCount() - timer);
157             if (ok)
158                 rectangle(frame, bbox,
159                       Scalar( 255, 0, 0 ), 2, 1 );
160             else
161                 putText(frame, "Tracking failure detected",
```

```
                    Point(100,80),
                    FONT_HERSHEY_SIMPLEX, 0.75,
                    Scalar(0,0,255),2);

            putText(frame, trackerType + " Tracker",
                Point(100,20),
                FONT_HERSHEY_SIMPLEX, 0.75,
                Scalar(50,170,50),2);

            putText(frame, "FPS : " + SSTR(int(fps)),
                Point(100,50),
                FONT_HERSHEY_SIMPLEX, 0.75,
                Scalar(50,170,50), 2);

            imshow("Tracking", frame);

            // Exit if ESC pressed.
            int k = waitKey(1);
            if(k == 27) break;
        } // end while
    } // end else
} // end main
```

B VOT connection implementation

Listing B.1: The implementation connection of trackers in language C++.

```
#include <opencv2/opencv.hpp>
#include <opencv2/tracking/tracking.hpp>
#include <opencv2/core/ocl.hpp>

#include <iostream>
#include <stdio.h>

#include "vot.h"

int main(int argc, char* argv[])
{
    cv::Ptr<cv::Tracker> tracker;
    tracker = cv::TRACKER::create();
    VOT vot;

    cv::Rect init, rect;
    init << vot.region();
    cv::Mat image = cv::imread(vot.frame());
    tracker->init(image, init);

    while (!vot.end()) {

        string imagepath = vot.frame();
        if (imagepath.empty()) break;

        cv::Mat image = cv::imread(imagepath);
        tracker->update(image, rect);
        cv::Rect rep(rect);
        vot.report(rep);

    }

    return 0;
}
```

C Results VOT2014 on mobile workstation

Tab. C.1: Experiment overview on mobile workstation

Experiment	baseline					region_noise				
	A-R rank				Speed	A-R rank				Speed
	A-Rank	R-Rank	Overlap	Failures	FPS	A-Rank	R-Rank	Overlap	Failures	FPS
OAB	7.16	5.00	0.46	37.75	43.3	6.08	4.75	0.45	37.62	42.4
MIL	6.41	**3.41**	0.46	**27.80**	16.4	6.00	**3.00**	0.45	**26.97**	16.6
MF	5.82	5.41	0.48	73.39	**291.7**	5.88	5.33	0.43	73.74	**292.0**
KCF	**3.05**	8.33	**0.58**	227.61	**143.3**	**3.64**	8.50	**0.55**	214.87	**140.4**
CSRT	**2.08**	**2.13**	**0.60**	**16.15**	40.8	**2.33**	**2.00**	**0.54**	**14.76**	40.0
MOSSE	5.76	8.00	0.52	246.28	**392.3**	5.80	7.75	0.47	202.65	**392.3**
GOTURN_O	**2.25**	**1.66**	**0.60**	**15.86**	17.0	**2.30**	**1.75**	**0.58**	**16.44**	17.1
GOTURN	6.59	3.69	0.43	36.48	12.5	6.20	4.41	0.41	34.83	12.2
TLD	6.00	7.33	0.45	118.75	34.7	6.63	7.50	0.41	119.62	28.4

Tab. C.2: Raw FPS results from baseline experiment on mobile workstation

	ball	basketball	bicycle	bolt	car	david	diving	drunk	fernando	fish1	fish2	gymnastics	hand1
OAB	43.7	35.7	65.7	39.5	68.3	29.2	40.3	27.5	24.4	51.5	35.4	31.2	39.4
MIL	15.4	15.7	15.8	14.4	15.6	17.6	18.5	18.8	16.0	16.2	16.7	16.7	16.1
MF	**411.4**	**184.3**	**309.3**	**129.7**	**235.5**	**369.5**	**349.9**	**285.7**	**151.5**	**275.8**	**183.2**	**428.7**	**342.4**
KCF	**189.2**	**127.6**	**223.1**	**115.9**	**135.7**	**132.2**	**155.6**	**87.6**	**56.9**	**223.9**	**123.4**	**110.3**	**134.9**
CSRT	46.5	35.9	59.3	36.1	50.3	39.6	35.1	35.6	29.6	44.3	36.0	40.0	39.3
MOSSE	**503.3**	**279.2**	**437.5**	**135.9**	**375.3**	**423.6**	**417.2**	**392.9**	**204.0**	**349.7**	**251.1**	**402.6**	**440.4**
GOTURN_O	17.2	16.9	17.1	16.2	17.2	17.2	17.1	17.1	16.5	17.1	16.8	17.1	17.1
GOTURN	14.5	14.2	14.6	13.7	14.3	-	10.6	-	10.6	10.7	11.3	12.6	10.7
TLD	49.5	28.6	49.0	33.3	38.1	39.6	58.5	34.1	26.0	30.6	21.2	57.0	43.1

	hand2	jogging	motocross	polarbear	skating	sphere	sunshade	surfing	torus	trellis	tunnel	woman	Average
OAB	35.1	50.0	21.3	48.4	40.8	27.2	55.8	82.5	43.4	48.4	45.5	51.5	43.3
MIL	16.7	16.8	16.0	16.7	15.5	17.1	16.0	16.0	16.5	16.8	17.4	16.0	16.4
MF	**343.4**	**344.8**	**203.1**	**215.0**	**190.6**	**265.0**	**318.9**	**441.9**	**311.0**	**394.1**	**260.7**	**347.9**	**291.7**
KCF	**152.9**	**144.3**	**69.1**	**110.2**	**98.2**	**89.1**	**191.6**	**312.6**	**169.3**	**158.2**	**113.4**	**157.0**	**143.3**
CSRT	38.9	40.7	28.8	42.3	33.4	35.7	40.5	65.2	42.1	43.0	41.8	39.2	40.8
MOSSE	**338.6**	**476.5**	**228.8**	**378.2**	**309.9**	**361.2**	**499.4**	**715.3**	**411.5**	**525.6**	**447.2**	**501.5**	**392.3**
GOTURN_O	17.1	17.2	16.9	17.1	16.9	16.9	17.2	17.3	17.1	17.3	17.2	17.2	17.0
GOTURN	9.2	13.2	12.7	13.0	12.0	12.8	11.0	13.4	11.5	13.1	13.1	12.7	12.5
TLD	46.6	37.0	-	12.7	26.4	24.3	29.4	30.4	28.9	30.5	23.0	35.1	34.7

Tab. C.3: Raw FPS results from region-noise experiment on mobile workstation

	ball	basketball	bicycle	bolt	car	david	diving	drunk	fernando	fish1	fish2	gymnastics	hand1
OAB	53.2	35.0	70.1	40.1	53.2	28.6	36.0	25.6	21.7	55.9	33.1	34.7	39.0
MIL	15.5	15.8	15.9	14.4	15.7	17.8	18.8	19.0	16.1	16.2	16.5	17.1	16.1
MF	**418.6**	**184.5**	**308.5**	**130.3**	**234.6**	**369.7**	**351.7**	**285.7**	**152.4**	**276.4**	**184.0**	**429.6**	**345.5**
KCF	**174.2**	**127.6**	**206.8**	**116.4**	**138.7**	**140.0**	**144.3**	**87.4**	**73.1**	**206.2**	**126.0**	**127.4**	**148.8**
CSRT	42.5	36.8	53.8	36.1	49.0	38.1	34.6	33.5	24.9	43.8	38.7	38.6	42.2
MOSSE	**507.7**	**279.8**	**454.2**	**144.2**	**372.2**	**413.0**	**414.2**	**376.4**	**195.0**	**370.4**	**264.1**	**388.7**	**432.9**
GOTURN_O	17.4	16.8	17.2	16.1	17.1	17.2	17.2	17.2	16.4	17.2	16.8	17.2	17.1
GOTURN	13.5	12.3	12.4	11.8	12.8	-	-	-	11.5	10.8	11.4	12.8	10.9
TLD	36.5	19.5	36.9	24.0	26.6	29.7	39.0	26.6	17.2	22.1	15.9	38.5	33.5

	hand2	jogging	motocross	polarbear	skating	sphere	sunshade	surfing	torus	trellis	tunnel	woman	Average
OAB	34.5	48.8	19.6	44.5	42.2	25.7	53.7	84.3	43.5	43.5	47.4	47.4	42.4
MIL	16.8	16.9	16.4	16.9	15.5	17.0	16.0	16.1	16.2	16.8	17.9	16.6	16.6
MF	**343.0**	**344.3**	**203.9**	**215.7**	**192.0**	**265.2**	**318.2**	**437.9**	**307.7**	**392.9**	**261.3**	**347.1**	**292.0**
KCF	**148.3**	**113.7**	**72.1**	**106.4**	**103.7**	**127.7**	**186.9**	**294.6**	**154.5**	**144.6**	**93.4**	**146.3**	**140.4**
CSRT	42.4	38.5	29.9	40.3	34.2	35.7	43.1	55.6	42.0	42.7	41.0	41.4	40.0
MOSSE	**345.5**	**481.8**	**236.7**	**376.3**	**304.8**	**356.8**	**489.4**	**738.9**	**407.1**	**520.9**	**430.6**	**506.4**	**392.3**
GOTURN_O	17.0	17.2	16.8	17.0	16.8	17.1	17.2	17.4	17.1	17.3	17.2	17.3	17.1
GOTURN	9.8	12.9	12.5	13.0	12.0	12.8	10.8	13.4	11.4	13.2	13.1	12.7	12.2
TLD	34.5	27.5	-	14.3	26.4	29.4	30.7	34.0	31.6	29.7	23.3	35.0	28.4

D Results VOT2014 on Raspberry Pi

Tab. D.1: Experiment overview on Raspberry Pi

Experiment	baseline					region_noise				
	A-R rank				Speed	A-R rank				Speed
	A-Rank	R-Rank	Overlap	Failures	FPS	A-Rank	R-Rank	Overlap	Failures	FPS
OAB	5.30	3.41	0.46	37.75	5.13	4.66	3.00	0.44	36.89	5.0
MIL	4.75	1.91	0.46	26.51	3.84	4.41	2.08	0.45	27.55	3.9
MF	4.11	3.66	0.49	71.88	36.7	4.88	3.75	0.43	74.82	36.2
KCF	2.41	6.33	0.58	226.86	15.1	2.47	6.50	0.54	211.92	14.3
CSRT	1.75	1.33	0.59	17.04	4.2	1.75	1.33	0.55	14.75	4.1
MOSSE	4.60	6.00	0.50	241.06	97.9	4.37	5.83	0.47	198.70	93.8
GOTURN	5.55	2.83	0.44	36.47	0.5	5.16	3.16	0.44	37.62	0.5
TLD	5.29	5.33	0.45	117.92	2.7	5.50	5.50	0.41	119.83	4.0

Tab. D.2: Raw FPS results from baseline experiment on Raspberry Pi

	ball	basketball	bicycle	bolt	car	david	diving	drunk	fernando	fish1	fish2	gymnastics	hand1
	ball	basketball	bicycle	bolt	car	david	diving	drunk	fernando	fish1	fish2	gymnastics	hand1
OAB	6.8	4.1	7.7	4.7	8.5	3.4	4.6	3.2	2.8	5.9	3.9	3.5	4.4
MIL	4.1	3.6	4.1	2.7	3.8	3.8	4.4	3.5	3.1	3.8	3.5	3.8	3.8
MF	56.7	14.9	41.3	13.3	20.5	56.0	50.7	25.1	12.4	30.1	15.3	77.3	57.5
KCF	22.1	12.6	26.3	15.8	13.1	13.7	16.9	5.3	2.3	29.4	12.4	10.5	13.7
CSRT	5.1	3.9	7.0	3.9	5.2	4.1	3.7	3.8	3.1	3.4	3.7	4.3	4.2
MOSSE	171.5	49.0	125.9	35.5	70.7	84.9	82.2	70.8	30.0	121.6	56.8	95.1	111.5
GOTURN	0.4	0.5	0.5	0.5	0.5	0.5	0.4	-	0.5	0.4	0.4	0.5	0.4
TLD	3.2	1.8	3.6	2.4	2.2	2.9	4.9	2.0	1.6	2.0	1.4	4.1	3.1

	hand2	jogging	motocross	polarbear	skating	sphere	sunshade	surfing	torus	trellis	tunnel	woman	Average
OAB	3.7	6.0	2.5	5.4	4.6	3.2	6.6	10.0	4.8	5.5	5.3	6.0	5.1
MIL	3.9	3.6	3.8	3.4	3.7	3.8	4.1	4.2	4.1	4.1	3.9	4.1	3.8
MF	62.0	35.8	17.2	17.7	17.0	25.4	37.0	65.6	52.3	56.7	22.4	38.0	36.7
KCF	15.7	14.9	3.4	10.0	6.8	8.4	22.4	40.7	19.3	16.2	10.2	15.9	15.1
CSRT	3.8	4.2	2.9	4.1	3.6	3.8	3.7	7.3	3.7	4.4	4.3	4.5	4.2
MOSSE	140.2	134.0	36.3	74.0	58.3	82.6	127.9	219.6	127.2	114.5	93.0	134.0	97.9
GOTURN	0.3	0.5	0.5	0.5	0.5	0.5	0.4	0.5	0.4	0.5	0.5	0.5	0.5
TLD	4.2	2.2	-	1.0	2.3	2.0	2.5	2.5	3.5	3.2	1.8	3.0	2.7

Tab. D.3: Raw FPS results from region-noise experiment on Raspberry Pi

	ball	basketball	bicycle	bolt	car	david	diving	drunk	fernando	fish1	fish2	gymnastics	hand1
OAB	6.5	3.5	8.2	4.8	7.5	3.3	4.2	3.0	2.5	6.4	3.7	4.1	4.5
MIL	4.1	3.6	4.3	3.8	4.0	4.0	4.3	3.8	3.5	3.8	3.6	4.0	4.0
MF	61.6	14.9	50.6	13.2	21.4	55.9	50.3	25.0	13.1	31.1	15.1	72.3	54.3
KCF	18.6	12.3	25.1	15.8	10.9	12.8	11.5	5.0	4.3	26.0	12.7	10.9	15.2
CSRT	4.1	3.8	5.3	3.8	5.5	3.9	3.6	3.6	2.7	4.0	3.8	3.8	4.3
MOSSE	158.1	50.4	119.5	31.7	81.0	86.1	81.3	65.8	24.6	114.0	50.2	99.0	109.4
GOTURN	0.5	0.5	0.5	0.5	0.5	0.5	0.4	-	0.5	0.4	0.4	0.5	0.4
TLD	3.2	1.8	3.4	2.3	2.1	4.5	6.6	3.1	2.3	3.6	2.1	6.8	5.7

	hand2	jogging	motocross	polarbear	skating	sphere	sunshade	surfing	torus	trellis	tunnel	woman	Average
OAB	3.8	5.7	2.2	5.2	4.8	3.1	6.0	9.8	4.9	5.1	5.8	5.7	5.0
MIL	4.0	4.1	3.9	3.8	3.8	3.9	4.1	4.2	4.1	4.1	3.8	4.0	3.9
MF	57.6	37.0	17.3	17.3	16.2	22.3	34.6	57.8	48.3	56.9	23.6	38.1	36.2
KCF	15.4	10.6	4.8	9.6	9.7	12.6	21.5	39.6	17.3	14.0	8.7	14.0	14.3
CSRT	4.4	4.1	3.2	4.1	3.5	3.8	4.4	5.3	4.3	4.3	4.3	4.4	4.1
MOSSE	132.8	119.5	34.3	73.1	56.2	72.8	127.8	212.7	116.9	115.3	90.9	120.9	93.8
GOTURN	0.3	0.5	0.5	0.5	0.5	0.5	0.4	0.5	0.4	0.5	0.5	0.5	0.5
TLD	6.0	4.2	-	1.7	3.4	4.2	4.5	5.2	5.6	5.3	3.5	5.7	4.0

E Content included CD

- `/` — root directory included CD
  - `Master_thesis.pdf` — pdf version of the thesis
  - `codes` — directory including all codes
    - `vot_bridge` — directory with VOT bridge
      - `CMakeLists.txt` — CMakefile for compilation VOT bridge
      - `vot.h` — VOT header
      - `opencv.cpp` — implementation of VOT bridge
      - `build` — directory with compiled code on Ubuntu 16.04 LTS
    - `Tracker` — directory with Tracker program
      - `CMakeLists.txt` — CMakefile for compilation Tracker program
      - `run.sh` — bash script for run Tracker program
      - `build` — directory with compiled code on Ubuntu 16.04 LTS
      - `src` — directory with source code
  - `reports_rasp` — directory with raw data for experiments on Raspberry
  - `reports_not` — directory with raw data for experiments on notebook